[Unit -1: Introduction]

Introduction to Cognitive Science (CSC-255)

Central Department of Computer Science & Information Technology
Tribhuvan University

Unit 1: Introduction

Cognitive*:* The earliest entries for the word "*cognitive*" in the *OED* take it to mean roughly *pertaining "to the action or process of knowing"*.

Cognitive science: the term

The term "*cognitive*" in "*cognitive science*" is "used for any kind of mental operation; cognitive process, or structure that can be studied in precise terms" (Lakoff and Johnson, 1999).

The term "*cognitive science"* was coined by Christopher Longuet-Higgins in his 1973 commentary on the *Lighthill report*, which concerned the then-current state of Artificial Intelligence research.

Cognitive Psychology:

Cognitive psychology is concerned with information processing, and includes a variety of processes such as attention, perception, learning, and memory. It is also concerned with the structures and representations involved in cognition. Cognitive psychology is one of the more recent additions to psychological research, having only developed as a separate area within the discipline since the late 1950s and early 1960s following the "cognitive revolution" initiated by Noam Chomsky's 1959 critique of behaviorism and empiricism more generally.

The core focus of cognitive psychology is on how people acquire, process and store information. There are numerous practical applications for cognitive research, such as ways to improve memory, how to increase decision-making accuracy, and how to structure educational curricula to enhance learning.

Cognitive science:

Cognitive science can be defined as the study of mind or the study of thought. We can also define it as the interdisciplinary study of *cognition*. **Cognition** includes mental states and processes such as thinking, remembering, language understanding and generation, visual and auditory perception, learning, consciousness, emotions, etc. It embraces multiple research disciplines, including *psychology*, *artificial intelligence*, *philosophy*, *neuroscience*, *linguistics*, *anthropology*, *sociology*, and *biology*. It relies on varying scientific methodology (e.g. *behavioral experimentation*, *computational simulations*, *neuro-imaging*, *statistical analyses*), and spans many levels of analysis of the mind (from low-level learning and decision mechanisms to high-level logic and planning, from neural circuitry to modular brain organization, etc.).

Some cognitive scientists limit their study to human cognition; other consider cognition independently of its implementation in humans or computers.

Cognitive science grew out of three developments: the invention of computers and the attempts to design programs that could do the kinds of tasks that humans do; the development of information processing psychology where the goal was to specify the internal processing involved in perception, language, memory, and thought; and the development of the theory of generative grammar and related offshoots in linguistics.

Cognitive science was a synthesis concerned with the kinds of knowledge that underlie human cognition, the details of human cognitive processing, and the computational modeling of those processes. There are five major topic areas in cognitive science: knowledge representation, language, learning, thinking, and perception.

Cognitive science differs from cognitive psychology in that algorithms that are intended to simulate human behavior are implemented or implementable on a computer

Cognitive science's approach to the study of mind is often contrasted with that of ***behaviorism***. The *behaviorist approach* to psychology seeks to describe and predict human behavior in terms of stimulus-response correlations, with no mention of unobservable mental states (including mental constructs such as symbols, ideas) or mental processes (such as thinking, planning, etc.) that might mediate these correlations. A behaviorist who would willing even to talk about the "mind" would view it as a "black box" that could only be understood in terms of its *input-output behavior*. Cognitive science in general seeks to understand human cognitive functions in terms of metal states and processes, i.e., in terms of *algorithm* that mediate between input and output.

The goal of cognitive science is to understand:

- the representations and processes in our minds that underwrite these capacities,
- how they are acquired, and how they develop, and
- how they are implemented in underlying hardware (biological or otherwise).

Stated more simply, the goal of cognitive science is to understand how the mind works.

Varieties of Cognitive Science:

Currently there are two major paradigms of computational cognitive science:

- Symbolic computational cognitive science
- Connectionist computational cognitive science

Symbolic computational cognitive science:

Here the concept is that the *mind* exists as a physically implemented "*symbol system*". A symbol system is any effectively computable procedure, i.e., a universal machine (which by Church's Thesis, could be a Turing machine, a recursive function, a general-purpose digital computer, etc.)

Connectionist computational cognitive science:

The "**connectionist**" (or "*neural network*", or "*parallel distributed processing*") approach to artificial intelligence and computational cognitive science can be seen as one way for a system to behave intelligently without being a "symbol system" and yet be computational. On this approach large numbers of very simple processors ("nodes") are connected in multiple ways by communication links of varying strengths. Input nodes receive information from the external world. The information is propagated along the links to and among intermediate nodes, finally reaching output nodes. Connectionist systems & techniques have been developed for learning features of natural language, for aspects of visual perception and for a number of other cognitive phenomena. The kind of information that the *symbol approach* would represent using various symbolic knowledge-representation techniques is, instead, "represented" by the strengths and connectivity patters of the links.

Cognitive science: a history

Like most approaches to the mind, cognitive science can be traced back to philosophical questions, especially about the nature of knowledge. For instance, Plato 's dialog Meno, where he investigates the source of knowledge, can be seen as a foundational text.

The modern culture of cognitive science can be traced back to the early cyberneticists; study of structure of regulatory system closely related to control theory & systems theory, in the **1930s and 1940s**, such as Warren McCulloch and Walter Pitts, who sought to understand the organizing principles of the mind. McCulloch and Pitts essentially invented the neural network, but did not have the computational tools to develop it into modern form.

Another precursor was the early development of the theory of computation and the digital computer in the **1940s and 1950s**. **Alan Turing and John von Neumann** were instrumental in these developments. The modern computer, or Von Neumann machine, would play a central role in cognitive science, both as a metaphor for the mind, and as a tool for investigation.

In 1959, Noam Chomsky published a scathing review of B. F. Skinner's book Verbal Behavior. At the time, **Skinner's behaviorist paradigm** dominated psychology: Most psychologists focused on functional relations between stimulus and response, without positing internal representations. Chomsky's work showed that in order to explain language, we needed a theory like his generative grammar; a particular approach to study of syntax in linguistic, which not only attributed internal representations but characterized their underlying order. This hugely successful theory would inspire much later cognitive science.

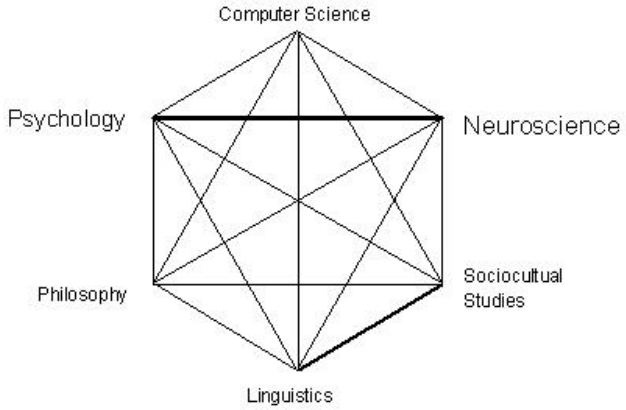
In the 1970s and early 1980s, much cognitive science research focused on the possibility of artificial intelligence. Researchers such as **Marvin Minsky** would write computer programs in languages such as LISP to attempt to formally characterize the steps that human beings went through, for instance, in making decisions and solving problems, in the hope of better understanding human thought, and also in the hope of creating artificial minds. This approach is known as "**symbolic AI**".

Eventually the limits of the symbolic AI research program became apparent. For instance, it seemed to be unrealistic to comprehensively list human knowledge in a form usable by a symbolic computer program. The late 80s and 90s saw the rise of neural networks and connectionism as a research paradigm. Under this point of view, often attributed to **James McClelland and David Rumelhart,** the mind could be characterized as a set of complex associations, represented as a layered network. Critics argue that there are some phenomena which are better captured by symbolic models, and that connectionist models are often so complex as to have little explanatory power.

Today a plurality of approaches exist, from connectionism, to a focus on dynamical systems models, to attempts to reintroduce symbolic models using tools from modern computer science such as **machine learning**. Machine learning is a scientific discipline that is concerned with the design and development of algorithms that allow computers to learn based on data, such as from sensor data or databases. A major focus of machine learning research is to automatically learn to recognize complex patterns.

Cognitive Science and Other Sciences

Cognitive science tends to view the world outside the mind much as other sciences do. Thus it too has an objective, observer-independent existence. The field is usually seen as compatible with the physical sciences, and uses the scientific method as well as simulation or modeling, often comparing the output of models with aspects of human behavior. Still, there is much disagreement about the exact relationship between cognitive science and other fields, and the interdisciplinary nature of cognitive science is largely both unrealized and circumscribed.



Philosophy:

Philosophy is the investigation of fundamental questions about the nature of knowledge, reality, and morals. It is the study of general and fundamental problems concerning matters such as *existence*, *knowledge*, *values*, *reason*, *mind*, and *language*. Philosophy is distinguished from other ways of addressing these questions by its critical, generally systematic approach.

Philosophy interfaces with cognitive science in three distinct but related areas. First, there is the usual set of issues that fall under the heading of philosophy of science (explanation, reduction, etc.), applied to the special case of cognitive science. Second, there is the endeavor of taking results from cognitive science as bearing upon traditional philosophical questions about the mind, such as the nature of mental representation, consciousness, free will, perception, emotions, memory, etc. Third, there is what might be called *theoretical cognitive science*, which is the attempt to construct the foundational theoretical framework and tools needed to get a science of the physical basis of the mind off the ground -- a task which naturally has one foot in cognitive science and the other in philosophy.

Psychological sciences: Psychology

Psychology is the study of mental activity. It incorporates the investigation of human mind and behavior & goes back at least to Plato and Aristotle.

Psychology is the science that investigates mental states directly. It uses generally empirical methods to investigate concrete mental states like *joy*, *fear* or *obsessions*. Psychology investigates the laws that bind these mental states to each other or with inputs and outputs to the human organism.

Psychology is now part of cognitive science, the interdisciplinary study of mind and intelligence, which also embraces the fields of neuroscience, artificial intelligence, linguistics, anthropology, and philosophy.

Biological sciences: Neuroscience

Neuroscience is a field of study which deals with the structure, function, development, genetics, biochemistry, physiology, pharmacology and pathology of the nervous system. The study of behavior and learning is also a division of neuroscience.

In cognitive science, it is very important to recognize the importance of neuroscience in contributing to our knowledge of human cognition. Cognitive scientists must have, at the very least, a basic understanding of, and appreciation for, neuroscientific principles. In order to develop accurate models, the basic neurophysiological and neuroanatomical properties must be taken into account.

Socio-cultural sciences: Sociology

Sociology is the scientific or systematic study of human societies. It is a branch of social science that uses various methods of empirical investigation and critical analysis to develop and refine a body of knowledge about human social structure and activity.

Linguistics

Linguistics is another discipline that is arguably wholly subsumed by cognitive science. After all, language is often held to be the "mirror of the mind"- the (physical) means for one mind to communicate its thoughts to another.

Linguistics is the scientific study of natural language. The study of language processing in cognitive science is closely tied to the field of linguistics. Linguistics was traditionally studied as a part of the humanities, including studies of history, art and literature. In the last fifty years or so, more and more researchers have studied knowledge and use of language as a cognitive phenomenon, the main problems being how knowledge of language can be acquired and used, and what precisely it consists of. Some of the driving research questions in studying how the brain processes language include:

(1) To what extent is linguistic knowledge innate or learned?

(2) Why is it more difficult for adults to acquire a second-language than it is for infants to acquire their first-language?

(3) How are humans able to understand novel sentences?

Computer Science: Artificial Intelligence

Artificial intelligence (AI) involves the study of cognitive phenomena in machines. One of the practical goals of AI is to implement aspects of human intelligence in computers. Textbooks define this field as "the study and design of intelligent agents". Computers are also widely used as a tool with which to study cognitive phenomena. Computational modeling uses simulations to study how human intelligence may be structured.

Given the computational view of cognitive science, it is arguable that all research in artificial intelligence is also research in cognitive science.

Mathematics

In mathematics, the theory of computation developed by Turing (1936) and others provided a theoretical framework for describing how states and processes interposed between input and output might be organized so as to execute a wide range of tasks and solve a wide range of problems. The framework of McCulluch and Pitts (1943) attempted to show how neuron-like units acting as and- and or- gates, etc., could be arranged so as to carry out complex computations. And while evidence that real neurons behave in this way was not forthcoming, it at least provided some hope for physiological vindication of such theories.

Descartes Mind Body Problem (Theory of dualism):

Dualism: The term ***dualism*** is the state of being dual, or having a twofold division. Dualism doctrine consists of two basic opposing elements. Generally it consists of any system which is founded on a double principle. In philosophy of mind, **dualism** is a set of views about the relationship between mind and matter, which begins with the claim that mental phenomena are, in some respects, non-physical.

A generally well-known version of **dualism** is attributed to **René Descartes** (1641), which holds that the mind is a nonphysical substance. Descartes was the first to clearly identify the *mind* with consciousness and self-awareness and to distinguish this from the brain, which was the seat of intelligence. Hence, he was the first to formulate the *mind-body problem* in the form in which it exists today.

The **mind-body problem** can be stated as, "*What is the basic relationship between the mental and the physical?*" For the sake of simplicity, we can state the problem in terms of mental and physical events: "*What is the basic relationship between mental events and physical events?*" It could also be stated in terms of the relation between mental and physical states and/or processes, or between the brain and consciousness.

The mind-body problem is that of stating the exact relation between the mind and the body, or, more narrowly, between the mind and the brain. Most of the theories of the mind-body relation exist also as metaphysical theories of reality as a whole. While debates over the mind-body problem can seem intractable, science offers at least two promising lines of research. On the one hand, parts of the mind-body problem arise in research in artificial intelligence and might be solved by a better understanding of the relations between hardware and software.

The famous **mind-body problem** has its origins in Descartes' conclusion that mind and body are really distinct. The crux of the difficulty lies in the claim that the **respective natures of mind and body are completely different** and, in some way, opposite from one another. On this account, **the mind is an entirely immaterial thing without any extension in it whatsoever**; and, conversely, **the body is an entirely material thing without any thinking in it at all**. This also means that each substance can have only its kind of modes. For instance, the mind can only have modes of understanding, will and, in some sense, sensation, while the body can only have modes of size, shape, motion, and quantity. But bodies cannot have modes of understanding or willing, since these are not ways of being extended; and minds cannot have modes of shape or motion, since these are not ways of thinking.

Descartes was aware that the positing of two distinct entities-a **rational mind** and a **mechanical body**-made implausible any explanation of their interaction. **How can an immaterial entity control, interact with, or react to a mechanical substance?** He made various stabs at solving this problem, none of them (as he knew) totally convincing. But in the process of trying to explain the interaction of mind and body, Descartes became in effect a physiologically oriented psychologist: he devised models of how mental states could exist

in a world of sensory experience-models featuring physical objects that had to be perceived and handled.

The basic steps in Descartes argument

- Reject any idea that can be doubted.
- Our senses deceive us (dreams).
- Our senses limit our knowledge (the wax example).
    - Knowledge is gained through the mind
- The only thing one cannot doubt is doubt itself.
- I doubt, therefore I think, therefore I am.

Descartes determined that the mind, an active reasoning entity, was the ultimate arbiter of truth. And he ultimately attributed ideas to innate rather than to experiential cause.

To further demonstrate the limitations of the senses, Descartes proceeds with what is known as the Wax Argument. He considers a piece of wax; his senses inform him that it has certain characteristics, such as shape, texture, size, color, smell, and so forth. When he brings the wax towards a flame, these characteristics change completely. However, it seems that it is still the same thing: it is still a piece of wax, even though the data of the senses inform him that all of its characteristics are different. Therefore, in order to properly grasp the nature of the wax, he cannot use the senses. He must use his mind. Descartes concludes: *And so something which I thought I was seeing with my eyes is in fact grasped solely by the faculty of judgment which is in my mind.*

Marr's Three Level of Information Processing:

In recent work in the theoretical foundations of cognitive science, it has become commonplace to separate three distinct levels of analysis of **information-processing systems**. David Marr (1982) has dubbed the three levels the *computational*, the *algorithmic*, and the *implementational*; Zenon Pylyshyn (1984) calls them the *semantic*, the *syntactic*, and the *physical*; and textbooks in cognitive psychology sometimes call them the levels of *content*, *form*, and *medium* (e.g. Glass, Holyoak, and Santa 1979).

David Marr presents his variant on the "three levels" story. His summary of "the three levels at which any machine carrying out an information-processing task must be understood":

- *Computational theory*: What is the goal of the computation, why is it appropriate, and what is the logic of the strategy by which it can be carried out? What is a concept? What does it mean to learn a concept successfully?

- *Representation and algorithm*: How can this computational theory be implemented? In particular, what is the representation for the input and output, and what is the algorithm for the transformation? How are objects and concepts represented? How much memory (space) and computation (time) does a learning algorithm require?

- *Hardware implementation*: How can the representation and algorithm be realized physically?

As an illustration, Marr applies this distinction to the levels of theorizing about a well-understood device: **a cash register**.

At the ***computational level***, "the level of *what* the device does and *why*", Marr tells us that "what it does is arithmetic, so our first task is to master the theory of addition".

But at the level of ***representation and algorithm***, which specifies the forms of the representations and the algorithms defined over them, "we might choose Arabic numerals for the representations, and for the algorithm we could follow the usual rules about adding the least significant digits first and `carrying' if the sum exceeds 9".

And, at the ***implementational level***, we face the question of how those symbols and processes are actually physically implemented; e.g., are the digits implemented as positions on a ten-notch metal wheel, or as binary coded decimal numbers implemented in the electrical states of digital logic circuitry?

Putting a closer look to Marr's, we might see the three perspectives of *algorithm*, *content of computation*, and *implementation* as having something like the following questions associated with them:

- *Format and algorithm*: What is the syntactic structure of the representations at this level, and what algorithms are used to transform them? What is the real structure of the virtual machine? What's the program? From this perspective, the questions are explicitly information-processing questions. Further, it's this level of functional decomposition of the system which specifies the level of organization with which we are currently concerned, and to which the other two perspectives are related.

- *Content, function, and interpretation*: What are the relational or global functional roles of the main processes described at this level? What tasks are being performed by these processes, and why? These are centrally questions about the interpretation and global function of the parts and procedures specified in our algorithmic analysis.

- *Implementation*: How are the primitives of the current level implemented? By another computationally characterized virtual machine? Directly in the hardware? How much decomposition (in terms of kinds of primitives, structures, abilities, etc.) is there between the current level and what is implementing it? How much of the work is done by the postulated primitives of this level as opposed to being done explicitly by the analyzed processes? The shift from algorithm to implementation is thus centrally one of levels of organization or functional decomposition; i.e. of what happens when we try to move down a level of organization.

Turing response to Descartes

Mind, in Descartes's view, is special, central to human existence, basically reliable. The mind stands apart from and operates independently of the human body, a totally different sort of entity. The body is best thought of as an **automaton**, which can be compared to the machines made by men. It is divisible into parts, and elements could be removed without altering anything fundamental. But even if one could design an automaton as complex as a human body, that automaton can never resemble the human mind, for the mind is unified and not decomposable. Moreover, unlike a human mind, a bodily machine could never use speech or other signs in placing its thoughts before other individuals. An automaton might parrot information, but "it never happens that it arranges its speech in various ways, in order to reply appropriately to everything that may be said in its presence, as even the lowest type of man can do" (quoted in Wilson 1969, p. 13S).

Turing devised the test in the 1950's, as a hypothetical test to determine when a machine had been imbued with sufficient intelligence to pass for human. In the test, a human judge is placed with two computer terminals, one connected to another human, and the other to a machine. The judge then converses with each terminal, and if he is unable to determine which terminal is connected to the machine, the machine is said to have attained similar intelligence to a human.

This test is often presented as a product of the 20th century. However, René Descartes' *Discourse on Method*, written in 1637, contains the following passage, which bears a fair resemblance to the Turing Test:

If there were machines which had the organs and the external shape of a monkey or of some other animal without reason, we would have no way of recognizing that they were not exactly the same nature as the animals; whereas, if there was a machine shaped like our bodies which imitated our actions as much as is morally possible, we would always have two very certain ways of recognizing that they were not, for all their resemblance, true human beings.

The first of these is that they would never be able to use words or other signs to make words as we do to declare our thoughts to others. For one can easily imagine a machine made in such a way that it expresses words, even that it expresses some words relevant to some physical actions which bring about some change in its organs (for example, if one touches it in some spot, the machine asks what it is that one wants to say to it; if in another spot, it cries that one has hurt it, and things like that), but one cannot imagine a machine that arranges words in various ways to reply to the sense of everything said in its presence, as the most stupid human beings are capable of doing.

It seems that Descartes was able to conceive not only of a machine that might mimic a human in form, but also in action and speech, and he reasoned that the best way to differentiate this machine from a human being would be to engage it in conversation, and observe whether it conversed naturally, in the manner of a human being, or whether the conversation would be driven solely by rote and logic.

Thus the Turing Machine, a mathematical automaton model, developed by Alan Turing in 1940 was addressed by the Descartes in his *Discourse on Method*.

In "Computer Technology," the section introducing Turing 1950, **Stuart Shieber**, well known to computational linguists for his research and to computer scientists, suggests that Turing played the role with respect to electronic computers that Descartes played with respect to mechanical devices, asking the same questions, only about different technology.

Application Related System in Cognitive Science

Major applications: decision making, education, human-machine interaction, intelligent systems. More,

- Neural Networks,
- Language Processing
- Machine Learning
- Machine Vision

[Unit -2: Introduction to Artificial Intelligence]

Introduction to Cognitive Science (CSC-255)

Central Department of Computer Science & Information Technology
Tribhuvan University

Unit 2: An Introduction to Artificial Intelligence

What is intelligence?

Intelligence is:

- the ability to reason
- the ability to understand
- the ability to create
- the ability to Learn from experience
- the ability to plan and execute complex tasks

What is Artificial Intelligence?

"Giving machines ability to perform tasks normally associated with *human* intelligence."

According to Barr and Feigenbaum:

> "Artificial Intelligence is the part of computer science concerned with designing intelligence computer systems, that is, systems that exhibit the characteristics we associate with intelligence in human behavior."

Different definitions of AI are given by different books/writers. These definitions can be divided into two dimensions.

Systems that think like humans	**Systems that think rationally**
"The exciting new effort to make computers think…..*machine with minds*, in the full and literal sense." (Haugeland, 1985) "[The automaton of] activities that we associate with human thinking, activities such as decision-making, problem solving, learning….." (Bellman, 1978)	"The study of mental faculties through the use of computational models." (Charniak and McDermott, 1985) "The study of the computations that make it possible to perceive, reason, and act." (Winston, 1992)
Systems that act like humans	**Systems that act rationally**
" The art of creating machines that perform functions that require intelligence when performed by people." (Kurzweil, 1990) "The study of how to make computer do things at which, at the moment, people are better." (Rich and Knight, 1991)	"Computational Intelligence is the study of the design of intelligent agents." (Poole *et al*., 1998) "AI… is concerned with intelligent behavior in artifacts." (Nilsson, 1998)

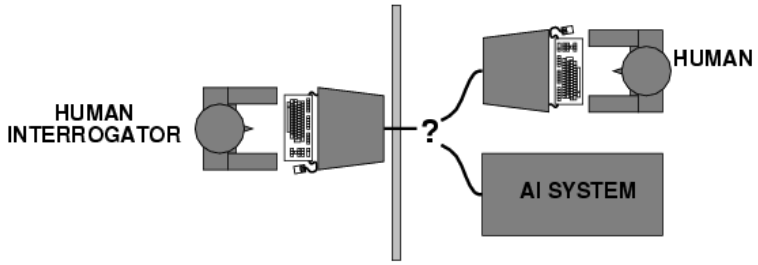
Top dimension is concerned with *thought processes and reasoning*, where as bottom dimension addresses the *behavior*.

The definition on the left measures the success in terms of fidelity of *human performance*, whereas definitions on the right measure an *ideal concept of intelligence*, which is called **rationality**.

Human-centered approaches must be an empirical science, involving hypothesis and experimental confirmation. A rationalist approach involves a combination of mathematics and engineering.

Acting Humanly: The Turing Test Approach

The **Turing test**, proposed by Alan Turing (1950) was designed to convince the people that whether a particular machine can think or not. He suggested a test based on indistinguishability from undeniably intelligent entities- human beings. **The test involves an interrogator who interacts with one human and one machine. Within a given time the interrogator has to find out which of the two the human is, and which one the machine**.



The computer passes the test if a human interrogator after posing some written questions, cannot tell whether the written response come from human or not.

To pass a Turing test, a computer must have following capabilities:

- Natural Language Processing: Must be able to communicate successfully in English
- Knowledge representation: To store what it knows and hears.
- Automated reasoning: Answer the Questions based on the stored information.
- Machine learning: Must be able to adapt in new circumstances.

Turing test avoid the physical interaction with human interrogator. Physical simulation of human beings is not necessary for testing the intelligence.

The total Turing test includes video signals and manipulation capability so that the interrogator can test the subject's perceptual abilities and object manipulation ability. To pass the total Turing test computer must have following additional capabilities:

- Computer Vision: To perceive objects
- Robotics: To manipulate objects and move

Thinking Humanly: Cognitive modeling approach

If we are going to say that a given program thinks like a human, we must have some way of determining how humans think. We need to get inside the actual workings of human minds. There are two ways to do this:

– **through introspection:** catch our thoughts while they go by
– **through psychological experiments.**

Once we have precise theory of mind, it is possible to express the theory as a computer program.

The field of cognitive science brings together computer models from AI and experimental techniques from psychology to try to construct precise and testable theories of the workings of the human mind.

Think rationally: The laws of thought approach

Aristotal was one of the first who attempt to codify the *right thinking* that is irrefutable reasoning process. He gave Syllogisms that always yielded correct conclusion when correct premises are given.

For example:

Ram is a man
All men are mortal
⇨ Ram is mortal

These law of thought were supposed to govern the operation of mind: This study initiated the field of logic. The logicist tradition in AI hopes to create intelligent systems using logic programming. However there are two obstacles to this approach. First, It is not easy to take informal knowledge and state in the formal terms required by logical notation, particularly when knowledge is not 100% certain. Second, solving problem principally is different from doing it in practice. Even problems with certain dozens of fact may exhaust the computational resources of any computer unless it has some guidance as which reasoning step to try first.

Acting Rationally: The rational Agent approach:

Agent is something that acts.

Computer agent is expected to have following attributes:

- Autonomous control
- Perceiving their environment
- Persisting over a prolonged period of time
- Adapting to change
- And capable of taking on another's goal

Rational behavior: doing the right thing.

The right thing: that which is expected to maximize goal achievement, given the available information.

Rational Agent is one that acts so as to achieve the best outcome or, when there is uncertainty, the best expected outcome.

In the "laws of thought" approach to AI, the emphasis was given to correct inferences. Making correct inferences is sometimes part of being a rational agent, because one way to act rationally is to reason logically to the conclusion and act on that conclusion. On the other hand, there are also some ways of acting rationally that cannot be said to involve inference. *For Example, recoiling from a hot stove is a reflex action that usually more successful than a slower action taken after careful deliberation.*

Advantages:

- It is more general than laws of thought approach, because correct inference is just one of several mechanisms for achieving rationality.
- It is more amenable to scientific development than are approaches based on human behavior or human thought because the standard of rationality is clearly defined and completely general.

Foundations of AI:

***Philosophy*:**
Logic, reasoning, mind as a physical system, foundations of learning, language and rationality.

- Where does knowledge come from?
- How does knowledge lead to action?
- How does mental mind arise from physical brain?
- Can formal rules be used to draw valid conclusions?

***Mathematics*:**
Formal representation and proof algorithms, computation, undecidability, intractability, probability.

- What are the formal rules to draw the valid conclusions?
- What can be computed?
- How do we reason with uncertain information?

***Psychology*:**
Adaptation, phenomena of perception and motor control.

- How humans and animals think and act?

***Economics*:**
Formal theory of rational decisions, game theory, operation research.

- How should we make decisions so as to maximize payoff?
- How should we do this when others may not go along?
- How should we do this when the payoff may be far in future?

***Linguistics*:**
Knowledge representation, grammar

- How does language relate to thought?

***Neuroscience*:**
Physical substrate for mental activities

- How do brains process information?

***Control theory*:**
Homeostatic systems, stability, optimal agent design

- How can artifacts operate under their own control?

Brief history of AI

- 1943: Warren Mc Culloch and Walter Pitts: a model of artificial boolean neurons to perform computations.
  - First steps toward connectionist computation and learning (Hebbian learning).
  - Marvin Minsky and Dann Edmonds (1951) constructed the first neural network computer

- 1950: Alan Turing's "Computing Machinery and Intelligence"
  - First complete vision of AI.

The birth of AI (1956):
- Dartmouth Workshop bringing together top minds on automata theory, neural nets and the study of intelligence.
  - Allen Newell and Herbert Simon: The logic theorist (first nonnumeric thinking program used for theorem proving)
  - For the next 20 years the field was dominated by these participants.

Great expectations (1952-1969):
- Newell and Simon introduced the General Problem Solver.
  - Imitation of human problem-solving

- Arthur Samuel (1952-) investigated game playing (checkers) with great success.
- John McCarthy(1958-) :
  - Inventor of Lisp (second-oldest high-level language)
  - Logic oriented, Advice Taker (separation between knowledge and reasoning)

- Marvin Minsky (1958 -)
  - Introduction of microworlds that appear to require intelligence to solve: e.g. blocks-world.
  - Anti-logic orientation, society of the mind.

Collapse in AI research (1966 - 1973):
- Progress was slower than expected.
  - Unrealistic predictions.
- Some systems lacked scalability.
  - Combinatorial explosion in search.
- Fundamental limitations on techniques and representations.
  - Minsky and Papert (1969) Perceptrons.

AI revival through knowledge-based systems (1969-1970):
- General-purpose vs. domain specific
  - E.g. the DENDRAL project (Buchanan et al. 1969)
    First successful knowledge intensive system.

- Expert systems
  - MYCIN to diagnose blood infections (Feigenbaum et al.)
    - Introduction of uncertainty in reasoning.

- Increase in knowledge representation research.
  - Logic, frames, semantic nets, …

AI becomes an industry (1980 - present):
- R1 at DEC (McDermott, 1982)
- Fifth generation project in Japan (1981)
- American response …

  Puts an end to the AI winter.

Connectionist revival (1986 - present): (Return of Neural Network):
- Parallel distributed processing (RumelHart and McClelland, 1986); backprop.

AI becomes a science (1987 - present):
- In speech recognition: hidden markov models
- In neural networks
- In uncertain reasoning and expert systems: Bayesian network formalism
- …

The emergence of intelligent agents (1995 - present):

- The whole agent problem:
  "How does an agent act/behave embedded in real environments with continuous sensory inputs"

Applications of AI

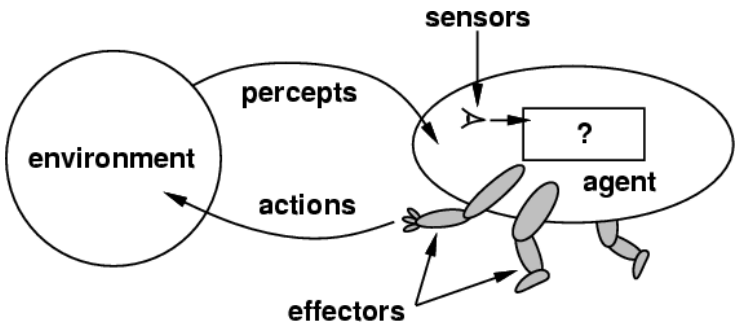
- Autonomous planning and scheduling
- Game playing
- Autonomous Control
- Diagnosis
- Logistics Planning
- Robotics
- Language understanding and problem solving

Intelligent Agents

An Intelligent Agent perceives it environment via sensors and acts rationally upon that environment with its effectors (actuators). Hence, an agent gets percepts one at a time, and maps this percept sequence to actions.

Properties of the agent

- Autonomous
- Interacts with other agents plus the environment
- Reactive to the environment
- Pro-active (goal- directed)



What do you mean, sensors/percepts and effectors/actions?

For Humans

- **Sensors**: Eyes (vision), ears (hearing), skin (touch), tongue (gestation), nose (olfaction), neuromuscular system (proprioception)
- **Percepts**:
  - At the lowest level – electrical signals from these sensors
  - After preprocessing – objects in the visual field (location, textures, colors, …), auditory streams (pitch, loudness, direction), …
- **Effectors**: limbs, digits, eyes, tongue, …..
- **Actions**: lift a finger, turn left, walk, run, carry an object, …

The Point: percepts and actions need to be carefully defined, possibly at different levels of abstraction

A more specific example: Automated taxi driving system

- **Percepts**: Video, sonar, speedometer, odometer, engine sensors, keyboard input, microphone, GPS, …
- **Actions**: Steer, accelerate, brake, horn, speak/display, …
- **Goals**: Maintain safety, reach destination, maximize profits (fuel, tire wear), obey laws, provide passenger comfort, …
- **Environment**: Urban streets, freeways, traffic, pedestrians, weather, customers, …

[Different aspects of driving may require different types of agent programs!]

Challenge!!

Compare Software with an agent
Compare Human with an agent

Percept: The Agents perceptual inputs at any given instant.
Percept Sequence: The complete history of everything the agent has ever perceived.

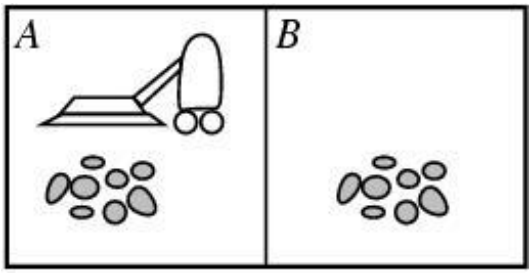
The *agent function* is mathematical concept that maps percept sequence to actions.

$$f : P^* \rightarrow A$$

The *agent function* will internally be represented by the *agent program.*

The agent program is concrete implementation of agent function it runs on the physical *architecture* to produce *f.*

The vacuum-cleaner world: Example of Agent



Environment: square A and B
Percepts: [location and content] *E.g. [A, Dirty]*
Actions: left, right, suck, and no-op

Percept sequence	Action
[A,Clean]	Right
[A, Dirty]	Suck
[B, Clean]	Left
[B, Dirty]	Suck
…….	……

The concept of rationality

A **rational agent** is one that does the right thing.

- Every entry in the table is filled out correctly.

What is the right thing?

- Right action is the one that will cause the agent to be most successful.

Therefore we need some way to measure success of an agent. Performance measures are the criterion for success of an agent behavior.

E.g., performance measure of a vacuum-cleaner agent could be amount of dirt cleaned up, amount of time taken, amount of electricity consumed, amount of noise generated, etc.

It is better to design Performance measure according to what is wanted in the environment instead of how the agents should behave.

It is not easy task to choose the performance measure of an agent. For example if the performance measure for automated vacuum cleaner is "The amount of dirt cleaned within a certain time" Then a rational agent can maximize this performance by cleaning up the dirt , then dumping it all on the floor, then cleaning it up again , and so on. Therefore "How clean the floor is" is better choice for performance measure of vacuum cleaner.

What is rational at a given time depends on four things:

- Performance measure,
- Prior environment knowledge,
- Actions,
- Percept sequence to date (sensors).
-

Definition: *A rational agent chooses whichever action maximizes the expected value of the performance measure given the percept sequence to date and prior environment knowledge.*

Environments

To design a rational agent we must specify its task environment. Task environment means: PEAS description of the environment:

- Performance
- Environment
- Actuators
- Sensors

Example: Fully automated taxi:

- PEAS description of the environment:

  Performance: Safety, destination, profits, legality, comfort

  Environment: Streets/freeways, other traffic, pedestrians, weather,, …

  Actuators: Steering, accelerating, brake, horn, speaker/display,…

  Sensors: Video, sonar, speedometer, engine sensors, keyboard, GPS, …

Knowledge Representation

Knowledge:

Knowledge is a theoretical or practical understanding of a subject or a domain. Knowledge is also the sum of what is currently known.

Knowledge is "the sum of what is known: the body of truth, information, and principles acquired by mankind." Or, "Knowledge is what I know, Information is what we know."

There are many other definitions such as:

- Knowledge is "information combined with experience, context, interpretation, and reflection. It is a high-value form of information that is ready to apply to decisions and actions." (T. Davenport et al., 1998)

- Knowledge is "human expertise stored in a person's mind, gained through experience, and interaction with the person's environment." (Sunasee and Sewery, 2002)

- Knowledge is "information evaluated and organized by the human mind so that it can be used purposefully, e.g., conclusions or explanations." (Rousa, 2002)

Knowledge consists of information that has been:

- interpreted,
- categorised,
- applied, experienced and revised.

In general, knowledge is more than just data, it consist of: facts, ideas, beliefs, heuristics, associations, rules, abstractions, relationships, customs.

Research literature classifies knowledge as follows:

Classification-based Knowledge	»	Ability to classify information
Decision-oriented Knowledge	»	Choosing the best option
Descriptive knowledge	»	State of some world (heuristic)
Procedural knowledge	»	How to do something
Reasoning knowledge	»	What conclusion is valid in what situation?
Assimilative knowledge	»	What its impact is?

Knowledge Representation

Knowledge representation (KR) is the study of how knowledge about the world can be represented and what kinds of reasoning can be done with that knowledge. Knowledge Representation is the method used to encode knowledge in Intelligent Systems.

Since knowledge is used to achieve intelligent behavior, the fundamental goal of knowledge representation is to represent knowledge in a manner as to facilitate inferencing (i.e. drawing conclusions) from knowledge. A successful representation of some knowledge must, then, be in a form that is *understandable* by humans, and must cause the system using the knowledge to *behave* as if it knows it.

Some issues that arise in knowledge representation from an AI perspective are:

- How do people represent knowledge?
- What is the nature of knowledge and how do we represent it?
- Should a representation scheme deal with a particular domain or should it be general purpose?
- How expressive is a representation scheme or formal language?
- Should the scheme be declarative or procedural?

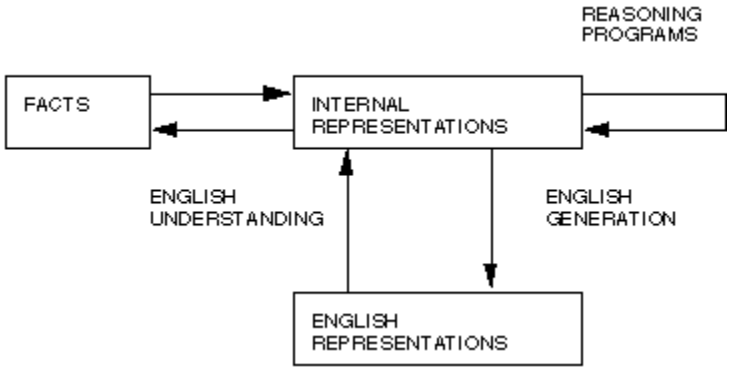


Fig: Two entities in Knowledge Representaion

For example: English or natural language is an obvious way of representing and handling facts. Logic enables us to consider the following fact: *spot is a dog* as *dog(spot)* We could then infer that all dogs have tails with: $\forall x$: *dog(x)* $\rightarrow$ *hasatail(x)* We can then deduce:

hasatail(Spot)

Using an appropriate backward mapping function the English sentence *Spot has a tail can be generated.*

Properties for Knowledge Representation Systems

The following properties should be possessed by a knowledge representation system.

Representational Adequacy
- the ability to represent the required knowledge;

Inferential Adequacy
- the ability to manipulate the knowledge represented to produce new knowledge corresponding to that inferred from the original;

Inferential Efficiency

- the ability to direct the inferential mechanisms into the most productive directions by storing appropriate guides;

Acquisitional Efficiency

- the ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

Approaches to Knowledge Representation

- **Rule-based**
  - IF <condition> THEN <conclusion>
- **Object-based**
  - Frames
  - Scripts
  - Semantic Networks
  - Object-Attribute-Value(O-A-V Triplets)
- **Example-based : Case-based Reasoning (CBR)**

These methods are interrelated and may be utilized in combination or individually within an expert system or other AI structure

Rule based approach:

Rule-based systems are used as a way to store and manipulate knowledge to interpret information in a useful way. In this approach, idea is to use production rules, sometimes called IF-THEN rules. The syntax structure is

IF <premise> THEN <action>

<premise> - is Boolean. The AND, and to a lesser degree OR and NOT, logical connectives are possible.
<action> - a series of statements

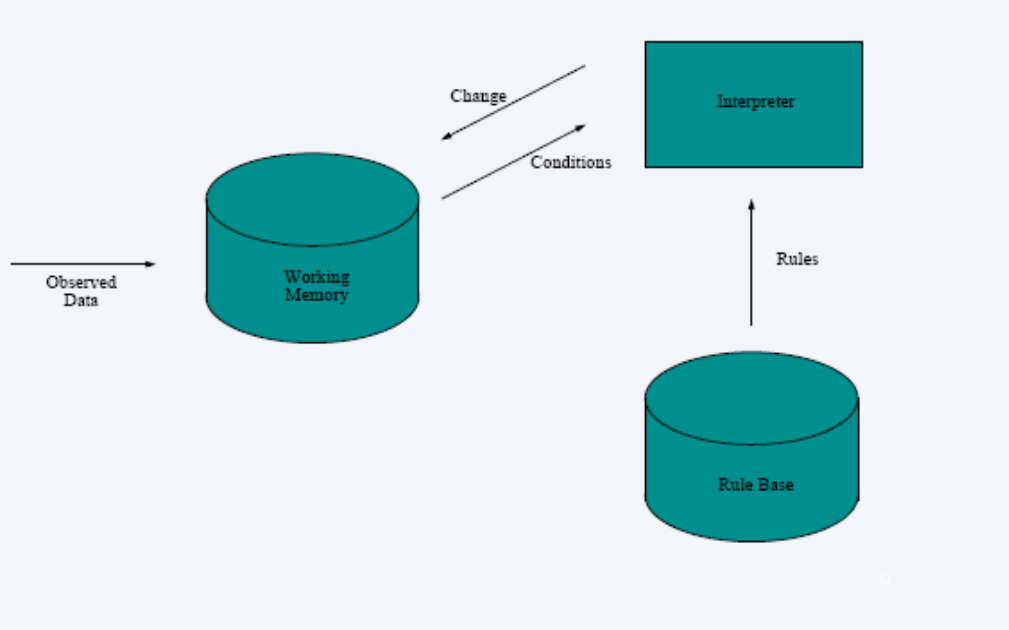
Notes:

- *The rule premise can consist of a series of clauses and is sometimes referred to as the antecedent*
- *The actions are sometimes referred to as the consequent*

A typical rule-based system has four basic components:

- A list of rules or **rule base**, which is a specific type of knowledge base.
- An **inference engine or semantic reasoner**, which infers information or takes action based on the interaction of input and the rule base.
- **Temporary working memory**.

- A **user interface** or other connection to the outside world through which input and output signals are received and sent.



Working Memory contains facts about the world and can be observed directly or derived from a rule. It contains temporary knowledge – knowledge about this problem-solving session. It may be modified by the rules.

It is traditionally stored as `<object, attribute, value>` triplet.

Rule Base contains rules; each rule is a step in a problem solving process. Rules are persistent knowledge about the domain. The rules are typically only modified from the outside of the system, e.g. by an expert on the domain.

The syntax isa `IF <conditions> THEN <actions>` format.

The conditions are matched to the working memory, and if they are fulfilled, the rule may be fired.

Actions can be:

- Adding fact(s) to the working memory.
- Removing fact(s) from the working memory
- Modifying fact(s) in the working memory.

The **Interpreter** operates on a cycle:

- *Retrieval*: Finds the rules that match the current Working Memory. These rules are the Conflict Set.
- *Refinement*: Prunes, reorders and resolves conflicts in the Conflict Set.

- *Execution:* Executes the actions of the rules in the Conflict Set. Applies the rule by performing the action.

Advantages of rule based approach:

- *Naturalness of Expression:* Expert knowledge can often been seen naturally as rules of thumb.

- *Modularity:* Rules are independent of each other – new rules can be added or revised later. Interpreter is independent from rules.

- *Restricted Syntax:* Allows construction of rules and consistency checking by other programs. Allows (fairly easy) rephrasing to natural language.

Disadvantages (or limitations)

- rule bases can be very large (thousands of rules)

- rules may not reflect the actual decision making

- the only structure in the KB is through the rule chaining

Examples: "If the patient has stiff neck, high fever and an headache, check for Brain Meningitis". Then it can be represented in rule based approach as:

IF <FEVER, OVER, 39> AND <NECK, STIFF, YES> AND <HEAD, PAIN, YES> THEN add(<PATIENT,DIAGNOSE, MENINGITIS>)

Example. Expert system for diagnosing car problems.

Rule 1: IF the engine is getting gas
AND the engine will turn over
THEN the problem is spark plugs

Rule 2: IF the engine does not turn over
AND the lights do not come on
THEN the problem is battery or cables.

Rule 3: IF the engine does not turn over
AND the lights do come on
THEN the problem is the starter motor.

Rule 4: IF there is gas in the fuel tank
AND there is gas in the carburettor
THEN the engine is getting gas

6

Object-based Approach: Frames

With this approach, knowledge may be represented in a data structure called a **frame**. A *frame* is a data structure containing typical knowledge about a concept or object (Marvin Minsky (mid 1970s)). A frame represents knowledge about real world things (or entities).

Each frame has **a name and slots**. **Slots** are the properties of the entity that has the name, and they have **values** or pointer to other frames (a table like data structure). A particular value may be:

- a default value
- an inherited value from a higher frame
- a procedure, called a daemon, to find a value
- a specific value, which might represent an exception.

When the slots of a frame are all filled, the frame is said to be **instantiated**, it now represents a specific entity of the type defined by the unfilled frame. Empty frames are sometimes called object prototypes

The idea of frame hierarchies is very similar to the idea of class hierarchies found in object-orientated programming. Frames are an application of the object-oriented approach to knowledge-based systems.

Disadvantages

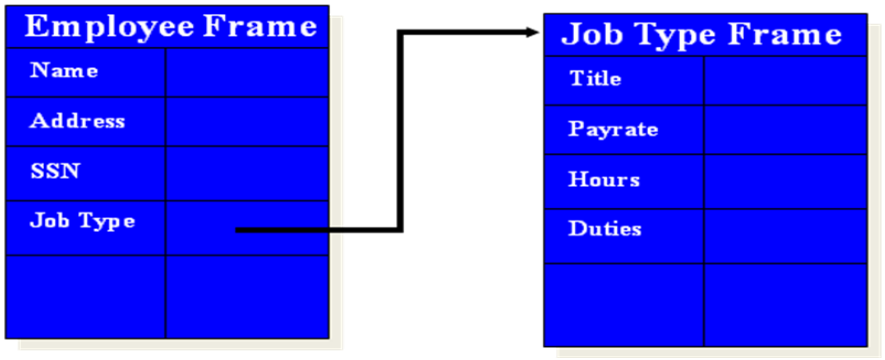
- complex
- reasoning (inferencing) is difficult
- explanation is difficult, expressive limitation

Advantages

- knowledge domain can be naturally structured [a similar motivation as for the O-O approach].
- easy to include the idea of default values, detect missing values, include specialised procedures and to add further slots to the frames

Examples:

(1.)



(2.)

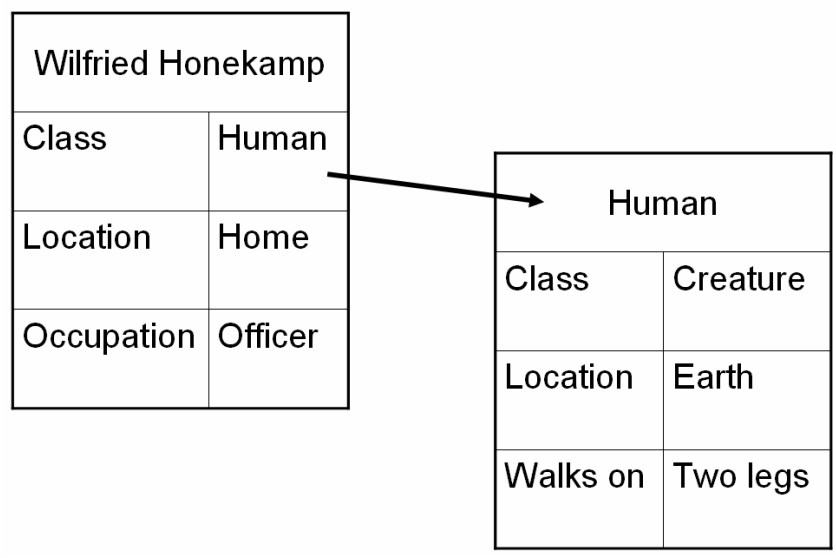


Fig: Two frames describing a human being

Object-based Approach: Semantic Network

Semantic Networks and Frames are called *Network Representations* or *Associative Representations*

Intuition: Knowledge is not a large collection of small pieces of knowledge but larger pieces that are highly interconnected. Logics and Rule-Based Systems seem to not have this property.

The meaning of concepts emerges from how it is connected to other concepts.

Semantic networks can
- show natural relationships between objects/concepts
- be used to represent declarative/descriptive knowledge

Knowledge is represented as a collection of concepts, represented by nodes. Thus, semantic networks are constructed using ***nodes*** linked by directional lines called ***arcs***

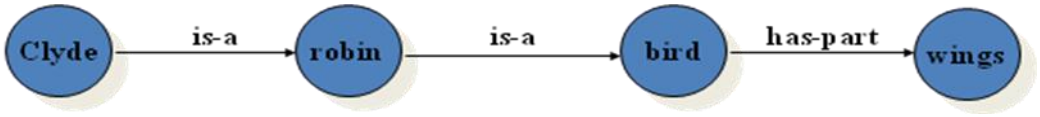
A node can represent a fact description
- physical object
- concept
- event

An arc (or link) represents relationships between nodes. There are some 'standard' relationship types
- 'Is-a' (instance relationship): represent class/instance relationships
- 'Has-a' (part-subpart relationship): identify property relationships

Semantic networks are mainly used as an aid to analysis to visually represent parts of the problem domain.

One feature of a semantic net is the ability to use the net to deduce new facts

- Begin with the fact that - all robins are birds
- If Clyde is the name of a particular pet robin then
- By following the is-a links it is easy to deduce that Clyde is a bird



Deduce: Robins have wings and Clyde has wings

More Examples:

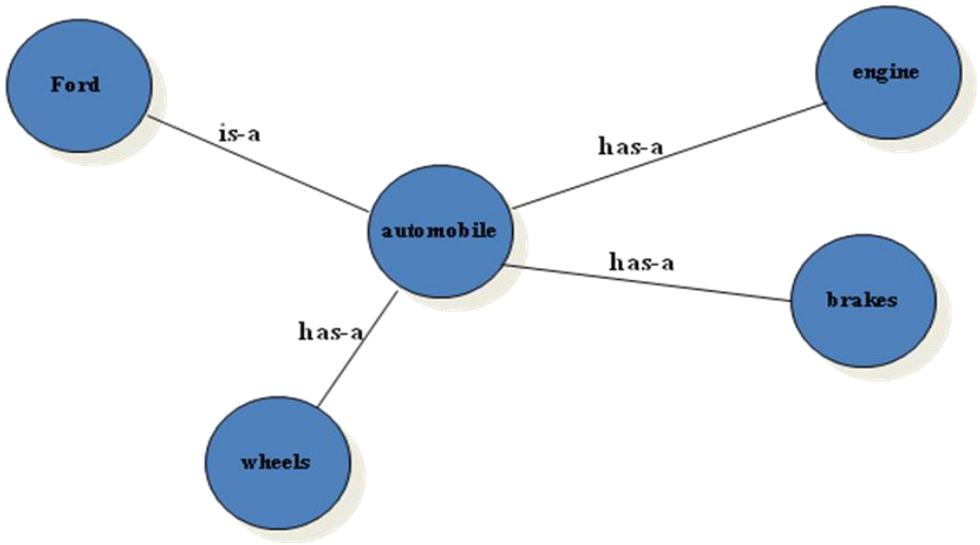


Fig: Automobile Semantic Net

Disadvantages of a semantic network

- incomplete (no explicit operational/procedural knowledge)
- no interpretation standard
- lack of standards, ambiguity in node/link descriptions
- not temporal (i.e. doesn't represent time or sequence)

Advantages of a semantic network

- Explicit and easy to understand.
- The net is its own index – quick inference possible.
- Supports default reasoning in finite time.
- Focus on bigger units of knowledge and interconnectedness.

O-A-V Triple:

Object – Attribute – Value (OAV) provides a particularly convenient way in which to represent certain facts and heuristic rules within KB. Each OAV triplet is present with specific entity, or object and a set of attributes with their values associated to every object.

Logic:

Logic is a formal language for representing knowledge such that conclusions can be drawn. **Logic** makes statements about the world which are true (or false) if the state of affairs it represents is the case (or not the case). Compared to natural languages (expressive but context sensitive) and programming languages (good for concrete data structures but not expressive) logic combines the advantages of natural languages and formal languages. Logic is concise, unambiguous, expressive, context insensitive, effective for inferences.

It has syntax, semantics, and proof theory.

Syntax: Describe possible configurations that constitute sentences.

Semantics: Determines what fact in the world, the sentence refers to i.e. the interpretation. Each sentence make claim about the world (meaning of sentence).Semantic property include truth and falsity.

Proof theory(Inference method): set of rules for generating new sentences that are necessarily true given that the old sentences are true.

We will consider two kinds of logic: **propositional logic** and **first-order logic** or more precisely first-order **predicate calculus**. Propositional logic is of limited expressiveness but is useful to introduce many of the concepts of logic's syntax, semantics and inference procedures.

Entailment:

Entailment means that one thing follows from another:

$KB \models \alpha$

Knowledge base KB entails sentence α if and only if α is true in all worlds where KB is true

E.g., $x + y = 4$ entails $4 = x + y$

Entailment is a relationship between sentences (i.e., syntax) that is based on semantics.

We can determine whether $S \models P$ by finding Truth Table for S and P, if any row of Truth Table where all formulae in S is true.

Example:

P	$P \rightarrow Q$	Q
True	True	True
True	False	False
False	True	True
False	True	False

Therefore {P, P→Q} |= Q. Here, only row where both P and P→Q are True, Q is also True. Here, S= (P, P→Q} and P= {Q}.

Models

Logicians typically think in terms of models, in place of "possible world", which are formally structured worlds with respect to which truth can be evaluated.

m is a model of a sentence α if α is true in m.

M(α) is the set of all models of α.

Propositional Logic:

Propositional logic represents knowledge/ information in terms of propositions. Prepositions are facts and non-facts that can be true or false. Propositions are expressed using ordinary declarative sentences. Propositional logic is the simplest logic.

Syntax:

The syntax of propositional logic defines the allowable sentences. The atomic sentences- the indivisible syntactic elements- consist of single proposition symbol. Each such symbol stands for a proposition that can be true or false. We use the symbols like P1, P2 to represent sentences.

The complex sentences are constructed from simpler sentences using logical connectives. There are five connectives in common use:

¬ (*negation*), ^ (*conjunction*), ∨ (*disjunction*), ⇒ (*implication*), ⇔ (*biconditional*)

The order of precedence in propositional logic is from (highest to lowest): ¬ , ^ , ∨, ⇒, ⇔.

Propositional logic is defined as:

If S is a sentence, ¬S is a sentence (*negation*)
If S1 and S2 are sentences, S1 ^ S2 is a sentence (*conjunction*)
If S1 and S2 are sentences, S1 ∨ S2 is a sentence (*disjunction*)
If S1 and S2 are sentences, S1 ⇒ S2 is a sentence (*implication*)
If S1 and S2 are sentences, S1 ⇔ S2 is a sentence (*biconditional*)

Formal grammar for propositional logic can be given as below:

Sentence	→ AutomicSentence | ComplexSentence
AutomicSentence	→ True | False | Symbol
Symbol	→ P | Q | R ……
ComplexSentence	→ ¬Sentence
	| (Sentence ^ Sentence)
	| (Sentence ∨ Sentence)
	| (Sentence ⇒ Sentence)
	| (Sentence ⇔ Sentence)

Semantics:

Each model specifies true/false for each proposition symbol

Rules for evaluating truth with respect to a model:

¬S is true if, S is false
S1 ^ S2 is true if, S1 is true and S2 is true
S1 ∨ S2 is true if, S1 is true or S2 is true
S1 ⇒ S2 is true if, S1 is false or S2 is true
S1 ⇔ S2 is true if, S1 ⇒ S2 is true and S2 ⇒ S1 is true

Truth Table showing the evaluation of semantics of complex sentences:

P	Q	¬P	P∧Q	P∨Q	P⇒Q	P⇔Q
false	false	true	false	false	true	true
false	true	true	false	true	true	false
true	false	false	false	true	false	false
true	true	false	true	true	true	true

Logical equivalence:

Two sentences α and ß are *logically equivalent* (α ≡ ß) iff true they are true inn same set of models or Two sentences α and ß are *logically equivalent* (α ≡ ß) iff α |= ß and ß |= α.

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) \quad \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) \quad \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) \quad \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) \quad \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) \quad \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) \quad \text{de Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) \quad \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \quad \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) \quad \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Validity:

A sentence is *valid* if it is true in all models,

e.g., *True*, A∨¬A, A ⇒ A, (A ∧ (A ⇒ B)) ⇒ B

Valid sentences are also known as tautologies. Every valid sentence is logically equivalent to True

Satisfiability:

A sentence is *satisfiable* if it is true in *some* model

- e.g., A ∨ B, C

A sentence is *unsatisfiable* if it is true in *no* models

- e.g., A¬∧A

Validity and satisfiablity are related concepts

- α is valid iff ¬α is unsatisfiable
- α is satisfiable iff ¬α is not valid

Satisfiability is connected to inference via the following:

- *KB* |= α if and only if (*KB* ∧ ¬α) is unsatisfiable

Inference rules in Propositional Logic

Modus Ponens

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

And-elimination

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

Monotonicity: the set of entailed sentences can only increase as information is added to the knowledge base.

For any sentence α and β if KB |= α then KB ∧ β |= α.

Resolution

Unit resolution rule:

Unit resolution rule takes a clause – a disjunction of literals – and a literal and produces a new clause. Single literal is also called unit clause.

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k}$$

Where l_i and m are complementary literals

Generalized resolution rule:
Generalized resolution rule takes two clauses of any length and produces a new clause as below.

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n}$$

For example:

$$\frac{\ell_1 \vee \ell_2, \qquad \neg\ell_2 \vee \ell_3}{\ell_1 \vee \ell_3}$$

Resolution Uses CNF (Conjunctive normal form)

- **Conjunction of disjunctions of literals (clauses)**

The resolution rule is sound:

- Only entailed sentences are derived

Resolution is complete in the sense that it can always be used to either confirm or refute a sentence (it can not be used to enumerate true sentences.)

Conversion to CNF

A sentence that is expressed as a conjunction of disjunctions of literals is said to be in conjunctive normal form (CNF). A sentence in CNF that contains only k literals per clause is said to be in k-CNF.

Algorithm:

Eliminate ↔rewriting P↔Q as (P→Q)∧(Q→P)

Eliminate →rewriting P→Q as ¬ P∨Q

Use De Morgan's 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)

Flatten nested clauses:

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

Example: Let's illustrate the conversion to CNF by using an example.

$B \Leftrightarrow (A \vee C)$

- Eliminate $\Leftrightarrow$, replacing $\alpha \Leftrightarrow$ ß with $(\alpha \Rightarrow$ ß$)\wedge($ß $\Rightarrow \alpha)$.
  - $(B \Rightarrow (A \vee C)) \wedge ((A \vee C) \Rightarrow B)$
- Eliminate $\Rightarrow$, replacing $\alpha \Rightarrow$ ß with $\neg \alpha \vee$ ß.
  - $(\neg B \vee A \vee C) \wedge (\neg(A \vee C) \vee B)$
- Move $\neg$ inwards using de Morgan's rules and double-negation:
  - $(\neg B \vee A \vee C) \wedge ((\neg A \wedge \neg C) \vee B)$
- Apply distributivity law ($\wedge$ over $\vee$) and flatten:
  - $(\neg B \vee A \vee C) \wedge (\neg A \vee B) \wedge (\neg C \vee B)$

Resolution algorithm

- Convert KB into CNF
- Add negation of sentence to be entailed into KB i.e. ($KB \wedge \neg\alpha$)
- Then apply resolution rule to resulting clauses.
- The process continues until:
  - There are no new clauses that can be added
    Hence *KB* **does not** entail α
  - Two clauses resolve to entail the empty clause.
    Hence *KB* **does** entail α

Example: Consider the knowledge base given as: $KB = (B \Leftrightarrow (A \vee C)) \wedge \neg B$
Prove that $\neg A$ can be inferred from above KB by using resolution.

Solution:

At first, convert KB into CNF

$B \Rightarrow (A \vee C)) \wedge ((A \vee C) \Rightarrow B) \wedge \neg B$

$(\neg B \vee A \vee C) \wedge (\neg(A \vee C) \vee B) \wedge \neg B$

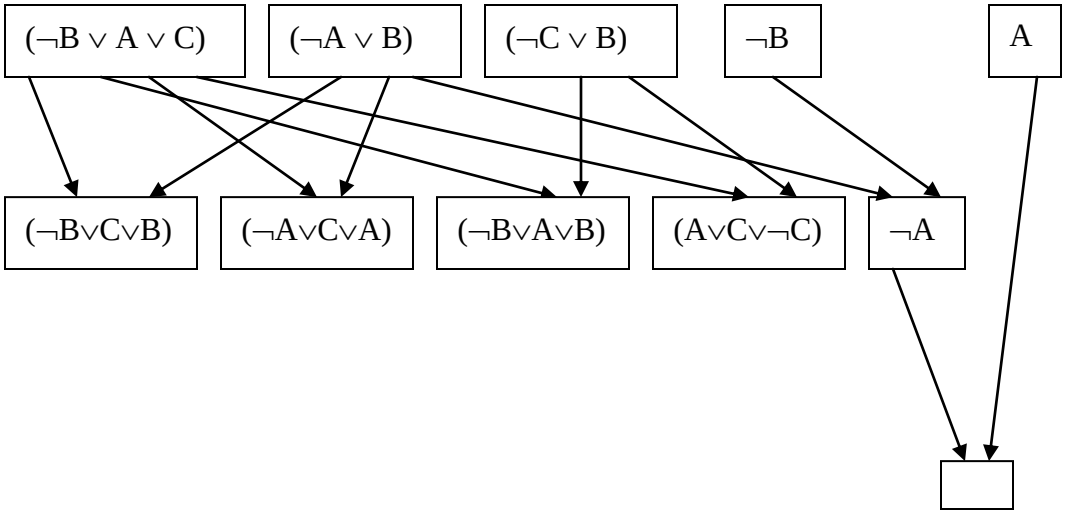
$(\neg B \vee A \vee C) \wedge ((\neg A \wedge \neg C) \vee B) \wedge \neg B$

$(\neg B \vee A \vee C) \wedge (\neg A \vee B) \wedge (\neg C \vee B) \wedge \neg B$

Add negation of sentence to be inferred from KB into KB

Now KB contains following sentences all in CNF
$(\neg B \vee A \vee C)$
$(\neg A \vee B)$
$(\neg C \vee B)$
$\neg B$
A (negation of conclusion to be proved)

Now use Resolution algorithm



Resolution: More Examples

1. KB= {(*G*∨*H*)→(¬ *J* ∧ ¬ *K*), *G*}. Show that KB ⊢ ¬ *J*

Solution:

Clausal form of (*G*∨*H*)→(¬ *J* ∧ ¬ *K*) is

{ ¬ *G*∨ ¬ *J*, ¬ *H* ∨ ¬ *J*, ¬ *G*∨ ¬ *K*, ¬ *H* ∨ ¬ *K*}

1. ¬ *G*∨ ¬ *J* [Premise]
2. ¬ *H* ∨ ¬ *J* [Premise]
3. ¬ *G*∨ ¬ *K* [Premise]
4. ¬ *H* ∨ ¬ *K* [Premise]
5. *G* [Premise]
6. *J* [¬ Conclusion]
7. ¬ *G* [1, 6 Resolution]
8. _ [5, 7 Resolution]

Hence KB entails ¬ *J*

2. KB= {*P*→ ¬ *Q*, ¬ *Q*→*R*}. Show that KB ⊢ *P*→*R*

Solution:

1. ¬ *P*∨ ¬ *Q* [Premise]
2. *Q*∨*R* [Premise]
3. *P* [¬ Conclusion]
4. ¬ *R* [¬ Conclusion]

5. $\neg Q$ [1, 3 Resolution]
6. R [2, 5 Resolution]
7. _ [4, 6 Resolution]

Hence, KB $\vdash P \rightarrow R$

3. $\vdash ((P \vee Q) \wedge \neg P) \rightarrow Q$

Clausal form of $\neg (((P \vee Q) \wedge \neg P) \rightarrow Q)$ is $\{P \vee Q, \neg P, \neg Q\}$

1. $P \vee Q$ [$\neg$ Conclusion]
2. $\neg P$ [$\neg$ Conclusion]
3. $\neg Q$ [$\neg$ Conclusion]
4. Q [1, 2 Resolution]
5. _ [3, 4 Resolution]

Forward and backward chaining

The completeness of resolution makes it a very important inference model. But in many practical situations full power of resolution is not needed. Real-world knowledge bases often contain only clauses of restricted kind called **Horn Clause.** A Horn clauses is disjunction of literals with at most one positive literal

Three important properties of Horn clause are:

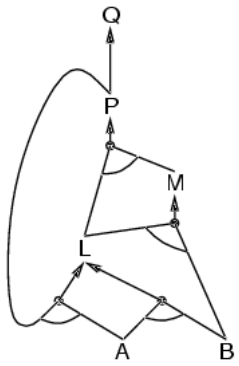
- ✓ Can be written as an implication
- ✓ Inference through forward chaining and backward chaining.
- ✓ Deciding entailment can be done in a time linear size of the knowledge base.

Forward chaining:

Idea: fire any rule whose premises are satisfied in the *KB*,

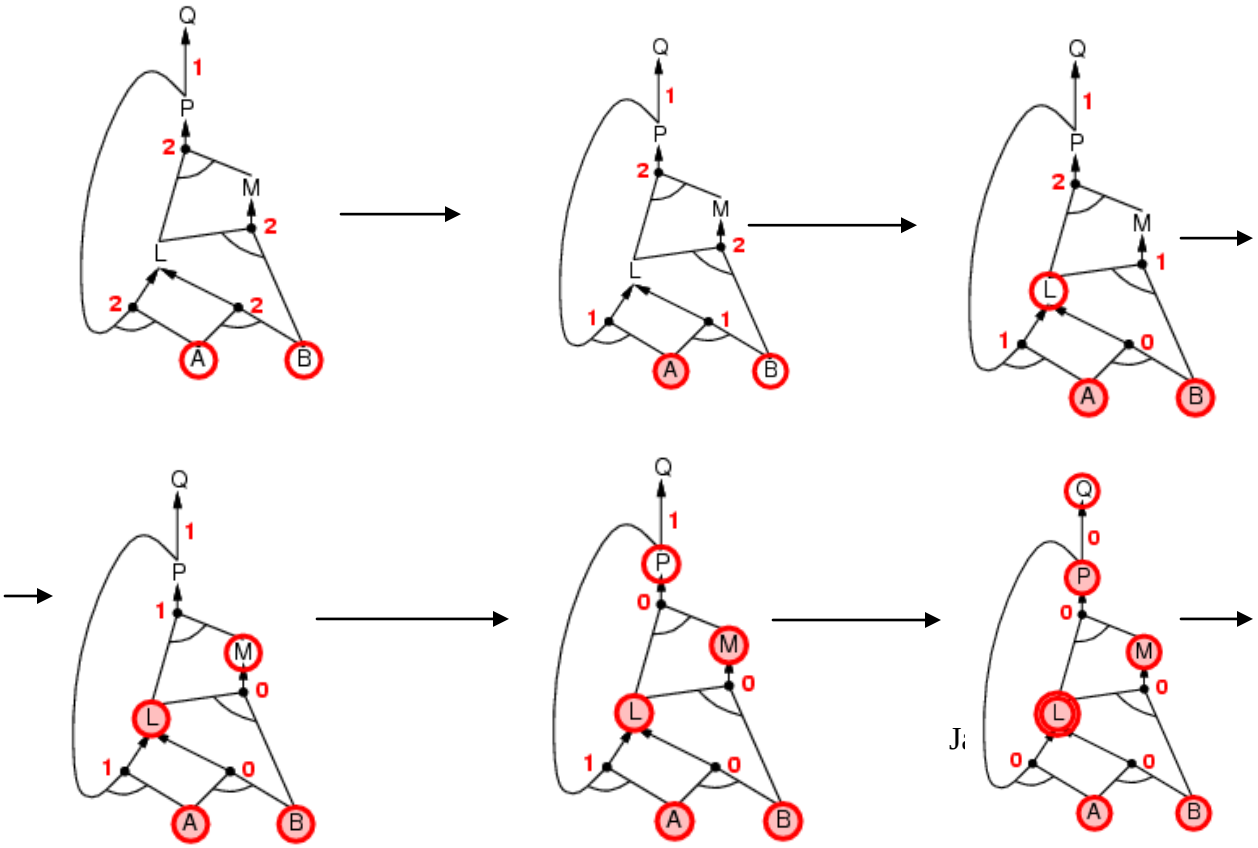
- add its conclusion to the *KB*, until query is found

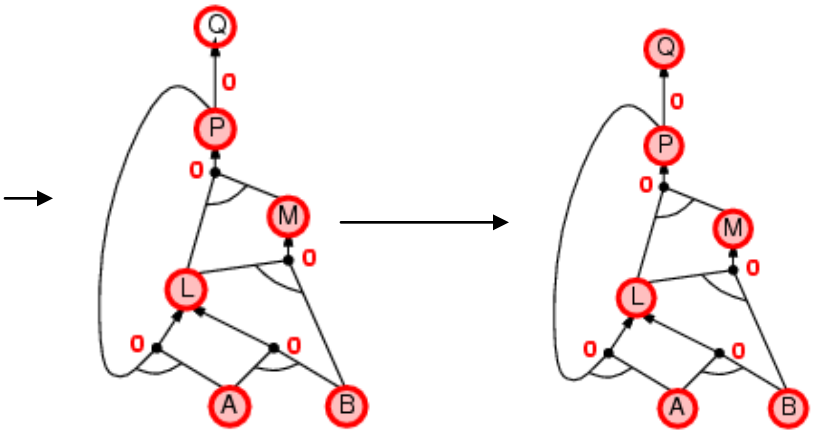
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



Prove that Q can be inferred from above KB

Solution:





Backward chaining:

Idea: work backwards from the query *q*: to prove *q* by BC,
Check if *q* is known already, or
Prove by BC all premises of some rule concluding *q*

For example, for above KB (as in forward chaining above)

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

Prove that Q can be inferred from above KB

Solution:
We know $P \Rightarrow Q$, try to prove P
$L \wedge M \Rightarrow P$
Try to prove L and M
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
Try to prove B, L and A and P
A and B is already known, since $A \wedge B \Rightarrow L$, L is also known
Since, $B \wedge L \Rightarrow M$, M is also known
Since, $L \wedge M \Rightarrow P$, p is known, hence the **proved.**

First-Order Logic

Pros and cons of propositional logic

- Propositional logic is declarative
- Propositional logic allows partial/disjunctive/negated information
  - (unlike most data structures and databases)
- Propositional logic is compositional:
  - meaning of $B \wedge P$ is derived from meaning of B and of P
- Meaning in propositional logic is context-independent
  - (unlike natural language, where meaning depends on context)
- Propositional logic has very limited expressive power
  - (unlike natural language)

Propositional logic assumes the world contains facts, whereas first-order logic (like natural language) assumes the world contains:

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

Logics in General

The primary difference between PL and FOPL is their ontological commitment:

Ontological Commitment: What exists in the world — TRUTH

- PL: facts hold or do not hold.
- FL : objects with relations between them that hold or do not hold

Another difference is:

Epistemological Commitment: What an agent believes about facts — BELIEF

Language	Ontological Commitment	Epistemological Commitment
Propositional logic	facts	true/false/unknown
First-order logic	facts, objects, relations	true/false/unknown
Temporal logic	facts, objects, relations, times	true/false/unknown
Probability theory	facts	degree of belief $\in [0,1]$
Fuzzy logic	degree of truth $\in [0,1]$	known interval value

FOPL: Syntax

Predicate Logic: Syntax

Sentence	→	*AtomicSentence*
	|	*(Sentence* `Connective` *Sentence)*
	|	`Quantifier` *Variable, ... Sentence*
	|	¬ *Sentence*
AtomicSentence	→	*Predicate(Term, …)* | *Term = Term*
Term	→	*Function(Term, …)* | *Constant* | *Variable*
Connective	→	∧ | ∨ | ⇒ | ⇔
Quantifier	→	∀ | ∃
Constant	→	*A, B, C, X_1, X_2, Jim, Jack*
Variable	→	*a, b, c, x_1, x_2, counter, position, ...*
Predicate	→	*Adjacent-To, Younger-Than, HasColor, ...*
Function	→	*Father-Of, Square-Position, Sqrt, Cosine*

ambiguities are resolved through precedence or parentheses

Representing knowledge in first-order logic

The objects from the real world are represented by constant symbols (a,b,c,...). For instance, the symbol "Tom" may represent a certain individual called Tom.

Properties of objects may be represented by predicates applied to those objects (P(a), ...): e.g "male(Tom)" represents that Tom is a male.

Relationships between objects are represented by predicates with more arguments: "father(Tom, Bob)" represents the fact that Tom is the father of Bob.

The value of a predicate is one of the boolean constants T (i.e. true) or F (i.e. false)."father(Tom, Bob) = T" means that the sentence "Tom is the father of Bob" is true. "father(Tom, Bob) = F" means that the sentence "Tom is the father of Bob" is false.

Besides constants, the arguments of the predicates may be functions (f,g,...) or variables (x,y,...).

Function symbols denote mappings from elements of a domain (or tuples of elements of domains) to elements of a domain. For instance, weight is a function that maps objects to their weight: weight (Tom) = 150.Therefore the predicate greater-than (weight (Bob), 100)

means that the weight of Bob is greater than 100. The arguments of a function may themselves be functions.

Variable symbols represent potentially any element of a domain and allow the formulation of general statements about the elements of the domain.

The quantifier's ∀ and ∃ are used to build new formulas from old ones.
"∃x P(x)" expresses that there is at least one element of the domain that makes P(x) true.
"∃x mother(x, Bob)" means that there is x such that x is mother of Bob or, otherwise stated, Bob has a mother.
"∀x P(x)" expresses that for all elements of the domain P(x) is true.

Quantifiers

Allows us to express properties of collections of objects instead of enumerating objects by name. Two quantifiers are:

Universal: "for all" ∀
Existential: "there exists" ∃

Universal quantification:

∀<*Variables*> <*sentence*>

Eg: Everyone at UAB is smart:
∀x At(x,UAB) ⇒ Smart(x)

∀x *P* is true in a model *m* iff *P* is true for all *x* in the model

Roughly speaking, equivalent to the conjunction of instantiations of *P*

At(KingJohn,UAB) ⇒ Smart(KingJohn) ∧ At(Richard,UAB) ⇒
Smart(Richard)∧At(UAB,UAB) ⇒ Smart(UAB)∧ ...

Typically, ⇒ is the main connective with ∀
- A universally quantifier is also equivalent to a set of implications over all objects

Common mistake: using ∧ as the main connective with ∀:
∀x At(x, UAB) ∧ Smart(x)
Means "Everyone is at UAB and everyone is smart"

Existential quantification

∃<*variables*> <*sentence*>
Someone at UAB is smart:
∃*x* At(x, UAB) ∧ Smart(x)

$\exists x$ *P* is true in a model *m* iff *P* is true for at least one *x* in the model

Roughly speaking, equivalent to the disjunction of instantiations of *P*

At(KingJohn,UAB) $\wedge$ Smart(KingJohn)$\vee$At(Richard,UAB) $\wedge$ Smart(Richard)
$\vee$At(UAB, UAB) $\wedge$ Smart(UAB) $\vee$ ...

Typically, $\wedge$ is the main connective with $\exists$

Common mistake: using $\Rightarrow$ as the main connective with $\exists$:
$\exists x$ At(x, UAB) $\Rightarrow$ Smart(x) is true even if there is anyone who is not at UAB!

FOPL: Semantic

An interpretation is required to give semantics to first-order logic. The interpretation is a non-empty "domain of discourse" (set of objects). The truth of any formula depends on the interpretation.

The interpretation provides, for each:
constant symbol an object in the domain
function symbols a function from domain tuples to the domain
predicate symbol a relation over the domain (a set of tuples)

Then we define:

universal quantifier $\forall xP(x)$ is True iff $P(a)$ is True for all assignments of domain elements *a* to *x*

existential quantifier $\exists xP(x)$ is True iff $P(a)$ is True for at least one assignment of domain element *a* to *x*

FOPL: Inference (Inference in first-order logic)

First order inference can be done by converting the knowledge base to PL and using propositional inference.

- How to convert universal quantifiers?
  - Replace variable by ground term.
- How to convert existential quantifiers?
  - Skolemization.

Universal instantiation (UI)

Substitute ground term (term without variables) for the variables.

For example consider the following KB
$\forall$ x King (x) $\wedge$ Greedy (x) $\Rightarrow$ Evil(x)

King (John)
Greedy (John)
Brother (Richard, John)

It's UI is:

King (John) $\wedge$ Greedy (John) $\Rightarrow$ Evil(John)
King (Richard) $\wedge$ Greedy (Richard) $\Rightarrow$ Evil(Richard)
King (John)
Greedy (John)
Brother (Richard, John)

Note: Remove universally quantified sentences after universal instantiation.

Existential instantiation (EI)

For any sentence α and variable v in that, introduce a constant that is not in the KB (called skolem constant) and substitute that constant for v.

E.g.: Consider the sentence, $\exists$ x Crown(x) $\wedge$ OnHead(x, John)

After EI,

Crown(C1) $\wedge$ OnHead(C1, John) where C1 is Skolem Constant.

Towards Resolution for FOPL:

- Based on resolution for propositional logic
- Extended syntax: allow variables and quantifiers
- Define "clausal form" for first-order logic formulae (CNF)
- Eliminate quantifiers from clausal forms
- Adapt resolution procedure to cope with variables (unification)

Conversion to CNF:

1. Eliminate implications and bi-implications as in propositional case
2. Move negations inward using De Morgan's laws

   plus rewriting $\neg \forall x P$ as $\exists x \neg P$ and $\neg \exists x P$ as $\forall x \neg P$
3. Eliminate double negations
4. Rename bound variables if necessary so each only occurs once

   e.g. $\forall x P(x) \vee \exists x Q(x)$ becomes $\forall x P(x) \vee \exists y Q(y)$
5. Use equivalences to move quantifiers to the left

   e.g. $\forall x P(x) \wedge Q$ becomes $\forall x\ (P(x) \wedge Q)$ where x is not in Q

   e.g. $\forall x P(x) \wedge \exists y Q(y)$ becomes $\forall x \exists y (P(x) \wedge Q(y))$
6. Skolemise (replace each existentially quantified variable by a **new** term)

   $\exists x P(x)$ becomes $P(a_0)$ using a Skolem constant a_0 since $\exists x$ occurs at the outermost level

   $\forall x \exists y P(x, y)$ becomes $P(x, f_0(x))$ using a Skolem function f_0 since $\exists y$ occurs within $\forall x$

7. The formula now has only universal quantifiers and all are at the left of the formula: drop them
8. Use distribution laws to get CNF and then clausal form

Example:

1.) $\forall x\,[\forall y P(x, y) \rightarrow \neg\, \forall y(Q(x, y) \rightarrow R(x, y))]$

Solution:

1. $\forall x\,[\neg\, \forall y P(x, y) \vee \neg\, \forall y(\neg\, Q(x, y) \vee R(x, y))]$
2, 3. $\forall x\,[\exists y \neg\, P(x, y) \vee \exists y(Q(x, y) \wedge \neg\, R(x, y))]$
4. $\forall x\,[\exists y \neg\, P(x, y) \vee \exists z\,(Q(x, z) \wedge \neg\, R(x, z))]$
5. $\forall x \exists y \exists z\,[\neg\, P(x, y) \vee (Q(x, z) \wedge \neg\, R(x, z))]$
6. $\forall x\,[\neg\, P(x, f(x)) \vee (Q(x, g(x)) \wedge \neg\, R(x, g(x)))]$
7. $\neg\, P(x, f(x)) \vee (Q(x, g(x)) \wedge \neg\, R(x, g(x)))$
8. $(\neg\, P(x, f(x)) \vee Q(x, g(x))) \wedge (\neg\, P(x, f(x)) \vee \neg\, R(x, g(x)))$
8. $\{\neg\, P(x, f(x)) \vee Q(x, g(x)),\ \neg\, P(x, f(x)) \vee \neg\, R(x, g(x))\}$

2.) $\neg\, \exists x \forall y \forall z\,((P(y) \vee Q(z)) \rightarrow (P(x) \vee Q(x)))$

Solution:

1. $\neg\, \exists x \forall y \forall z\,(\neg\, (P(y) \vee Q(z)) \vee P(x) \vee Q(x))$
2. $\forall x \neg\, \forall y \forall z\,(\neg\, (P(y) \vee Q(z)) \vee P(x) \vee Q(x))$
2. $\forall x \exists y \neg\, \forall z\,(\neg\, (P(y) \vee Q(z)) \vee P(x) \vee Q(x))$
2. $\forall x \exists y \exists z \neg\,(\neg\, (P(y) \vee Q(z)) \vee P(x) \vee Q(x))$
2. $\forall x \exists y \exists z\,((P(y) \vee Q(z)) \wedge \neg\, (P(x) \vee Q(x)))$
6. $\forall x\,((P(f(x)) \vee Q(g(x))) \wedge \neg\, P(x) \wedge \neg\, Q(x))$
7. $(P(f(x)) \vee Q(g(x))) \wedge \neg\, P(x) \wedge \neg\, Q(x)$
8. $\{P(f(x)) \vee Q(g(x)),\ \neg\, P(x),\ \neg\, Q(x)\}$

Unification:

A unifier of two atomic formulae is a substitution of terms **for variables** that makes them identical.

- Each variable has at most one associated term
- Substitutions are applied simultaneously

Unifier of $P(x, f(a), z)$ and $P(z, z, u)$: $\{x/ f(a), z/ f(a), u/ f(a)\}$

We can get the inference immediately if we can find a substitution α such that *King(x)* and *Greedy(x)* match *King(John)* and *Greedy(y)*

α = {x/John,y/John} works

Unify(α ,β) = θ if $\alpha\theta = \theta\beta$

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}}
Knows(John,x)	Knows(x,OJ)	{fail}

Last unification is failed due to overlap of variables. x can not take the values of John and OJ at the same time.

We can avoid this problem by renaming to avoid the name clashes (standardizing apart)

E.g.

Unify{Knows(John,x) Knows(z,OJ) } = {x/OJ, z/John}

Another complication:

To unify *Knows(John,x)* and *Knows(y,z),*

Unification of Knows*(John,x)* and *Knows(y,z)* gives α ={y/John, x/z } or α={y/John, x/John, z/John}

First unifier gives the result Knows(John,z) and second unifier gives the resultKnows(John, John). Second can be achieved from first by substituting john in place of z. The first unifier is more general than the second.

There is a single most general unifier (MGU) that is unique up to renaming of variables.

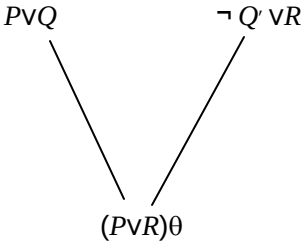
MGU = { y/John, x/z }

Towards Resolution for First-Order Logic

- Based on resolution for propositional logic
- Extended syntax: allow variables and quantifiers
- Define "clausal form" for first-order logic formulae
- Eliminate quantifiers from clausal forms
- Adapt resolution procedure to cope with variables (unification)

First-Order Resolution

For clauses $P \vee Q$ and $\neg Q' \vee R$ with Q, Q' atomic formulae



where θ is a most general unifier for Q and Q'

$(P \vee R)\theta$ is the resolvent of the two clauses

Applying Resolution Refutation

- Negate query to be proven (resolution is a refutation system)
- Convert knowledge base and negated query into CNF and extract clauses
- Repeatedly apply resolution to clauses or copies of clauses until either the empty clause (contradiction) is derived or no more clauses can be derived (a copy of a clause is the clause with all variables renamed)
- If the empty clause is derived, answer 'yes' (query follows from knowledge base), otherwise answer 'no' (query does not follow from knowledge base)

Resolution: Examples

1.) $\vdash \exists x\,(P(x) \rightarrow \forall x P(x))$

Solution:

Add negation of the conclusion and convert the predicate in to CNF:

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

1, 2. $\forall x \neg (\neg P(x) \vee \forall x P(x))$

2. $\forall x\,(\neg \neg P(x) \wedge \neg \forall x P(x))$

2, 3. $\forall x\,(P(x) \wedge \exists x \neg P(x))$

4. $\forall x\ (P(x) \wedge \exists y\ \neg P(y))$
5. $\forall x \exists y (P(x) \wedge \neg P(y))$
6. $\forall x\ (P(x) \wedge \neg P(f(x)))$
8. $P(x),\ \neg P(f(x))$

_____

Now, we can use resolution as;

1. $P(x)$ [¬ Conclusion]
2. $\neg P(f(y))$ [Copy of ¬ Conclusion]
3. _ [1, 2 Resolution $\{x/f(y)\}$]

2.) $\vdash \exists x \forall y \forall z\ ((P(y) \vee Q(z)) \rightarrow (P(x) \vee Q(x)))$

Solution:

1. $P(f(x)) \vee Q(g(x))$ [¬ Conclusion]
2. $\neg P(x)$ [¬ Conclusion]
3. $\neg Q(x)$ [¬ Conclusion]
4. $\neg P(y)$ [Copy of 2]
5. $Q(g(x))$ [1, 4 Resolution $\{y/f(x)\}$]
6. $\neg Q(z)$ [Copy of 3]
7. _ [5, 6 Resolution $\{z/g(x)\}$]

3.)

The following axioms describe the situation:

1. If the coin comes up heads, then I win.
2. If it comes up tails, then you lose.
3. If it does not come up heads, then it comes up tails.
4. if you lose, then I win.

Which may be represented as:

1. $H \rightarrow W(me)$ //H: heads , W: win
2. $T \rightarrow L(you)$ //T: tails, L: lose
3. $\neg H \rightarrow T$
4. $L(you) \rightarrow W(me)$

Next, our argument is converted to clause form

1. $\neg H \vee W(me)$
2. $\neg T \vee L(you)$
3. $H \vee T$
4. $\neg L(you) \vee W(me)$

Then, add the negation of the conclusion

5. $\neg W(me)$ //also in clause form

Finally, we attempt to obtain a contradiction

2,4	$\neg T \vee W(me)$	6
1,3	$T \vee W(me)$	7
6,7	$W(me)$	8
5,8	$\square$	**//contradiction!**

Hence W(me) //I win!!

Human Information Processing and Problem Solving

Human Information Processing:

The **information processing theory** approach to the study of cognitive development evolved out of the American experimental tradition in psychology. Information processing theorists proposed that like the computer, the human mind is a system that processes information through the application of logical rules and strategies. Like the computer, the mind has a limited capacity for the amount and nature of the information it can process.

Finally, just as the computer can be made into a better information processor by changes in its hardware (e.g., circuit boards and microchips) and its software (programming), so do children become more sophisticated thinkers through changes in their brains and sensory systems (hardware) and in the rules and strategies (software) that they learn.

Human information processing theory deals with how people receive, store, integrate, retrieve, and use information.

Since the first computers, psychologists have drawn parallels between computers and human thought. At its core are memory models. The memory model which dominated the 1970's and 80's is the three component information processing system of **Atkinson and Shiffrin** (1968, 1971) inspired by typical computer hardware architecture:

- Sensory Memory (STSS): Analogous to input devices such as a keyboard or more sophisticated devices like a voice recognition system
- Short Term Memory (STM) or working memory: Analogous to the CPU and it's random-access memory (RAM)
- Long Term Memory (LTM) : Analogous to a storage device like a hard disk

Principles of information processing approach

According to Huitt (2003), there are a few basic principles that most cognitive psychologists agree with:

- The mental system has limited capacities, i.e. bottlenecks in the flow and processing of information, occur at very specific points
- A control mechanism is required to oversee the encoding, transformation, processing, storage, retrieval and utilization of information. This control mechanism requires itself processing power and that varies in function of the difficulty of the task.
- There is a two-way flow of information. Sensory input is combined with information stored in memory in order to construct meaning.
- The human organism has been genetically prepared to process and organize information in specific ways.

Structure of the information-processing system

In the store model of the human information-processing system, information from the environment that we acquire through our senses enters the system through the sensory register.

- **The store model:** A model of information processing in which information is depicted as moving through a series of processing units — sensory register, short-term memory, long-term memory — in each of which it may be stored, either fleetingly or permanently.

- **Sensory register:** the mental processing unit that receives information from the environment and stores it momentarily.

- **Short-term memory:** the mental processing unit in which information may be stored temporarily; the work space of the mind, where a decision must be made to discard information or to transfer it to permanent storage, in long-term memory.

- **Long-term memory:** the encyclopedic mental processing unit in which information may be stored permanently and from which it may be later retrieved.

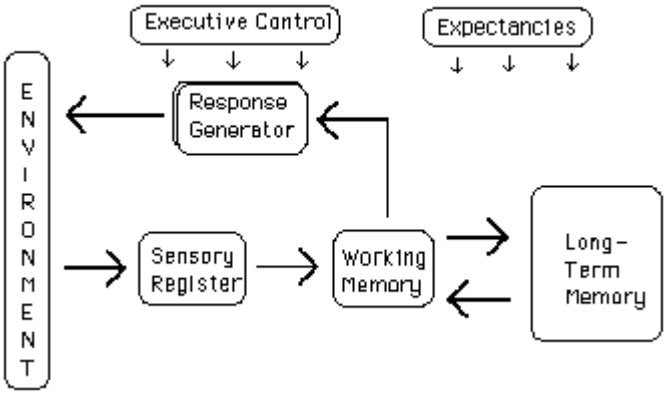


Fig: A model of human Information Processing

Problem Solving:

Problem solving, particularly in artificial intelligence, may be characterized as a systematic search through a range of possible actions in order to reach some predefined goal or solution. Problem-solving methods divide into special purpose and general purpose. A special-purpose method is tailor-made for a particular problem and often exploits very specific features of the situation in which the problem is embedded. In contrast, a general-purpose method is applicable to a wide variety of problems. One general-purpose technique used in AI is means-end analysis—a step-by-step, or incremental, reduction of the difference between the current state and the final goal.

Four general steps in problem solving:

- Goal formulation
  - What are the successful world states
- Problem formulation
  - What actions and states to consider given the goal
- Search
  - Determine the possible sequence of actions that lead to the states of known values and then choosing the best sequence.
- Execute
  - Give the solution perform the actions.

Problem formulation:

A problem is defined by:

- An initial state: State from which agent start
- Successor function: Description of possible actions available to the agent.
- Goal test: Determine whether the given state is goal state or not
- Path cost: Sum of cost of each path from initial state to the given state.

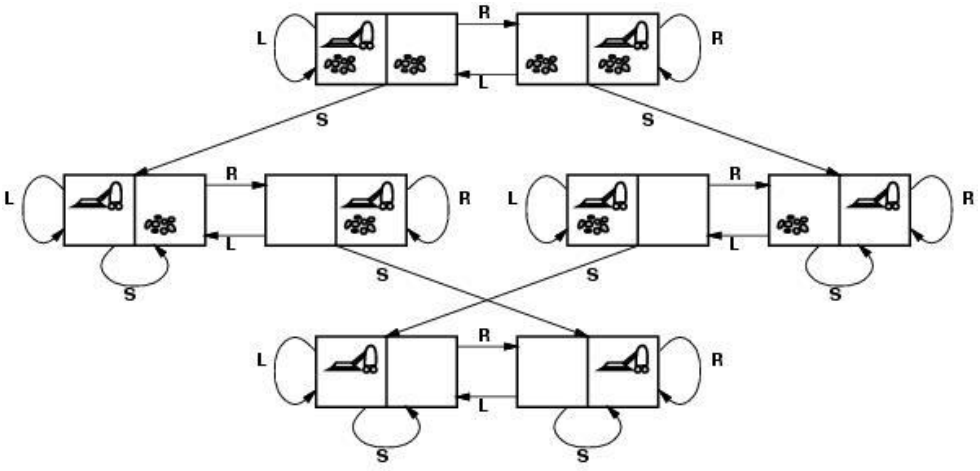
A solution is a sequence of actions from initial to goal state. Optimal solution has the lowest path cost.

State Space representation

The state space is commonly defined as a directed graph in which each node is a state and each arc represents the application of an operator transforming a state to a successor state.

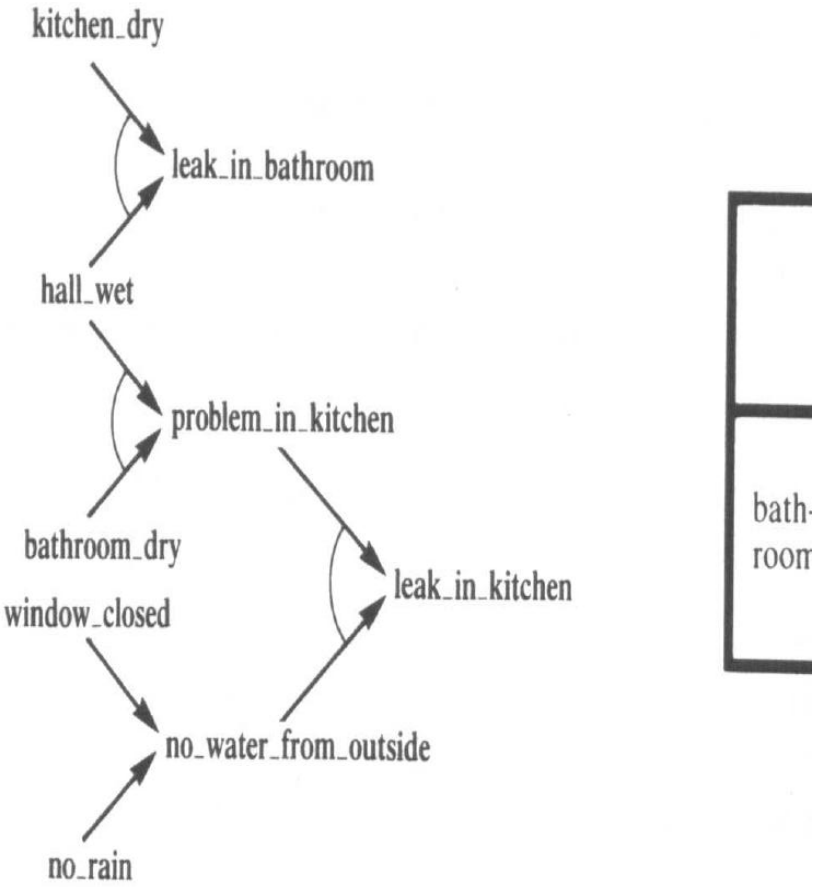
A **solution** is a path from the initial state to a goal state.

State Space representation of Vacuum World Problem:



States?? two locations with or without dirt: 2 x 2^2=8 states.
Initial state?? Any state can be initial
Actions?? {*Left, Right, Suck*}
Goal test?? Check whether squares are clean.
Path cost?? Number of actions to reach goal.

Water Leakage Problem:



If

hall

_wet and kitchen_dry
then
leak_in_bathroom
If
hall_wet and bathroom_dry
then
problem_in_kitchen
If
window_closed or no_rain
then
no_water_from_outside

Searching

A search problem

Figure below contains a representation of a map. The nodes represent cities, and the links represent direct road connections between cities. The number associated to a link represents the length of the corresponding road.

The search problem is to find a path from a city S to a city G

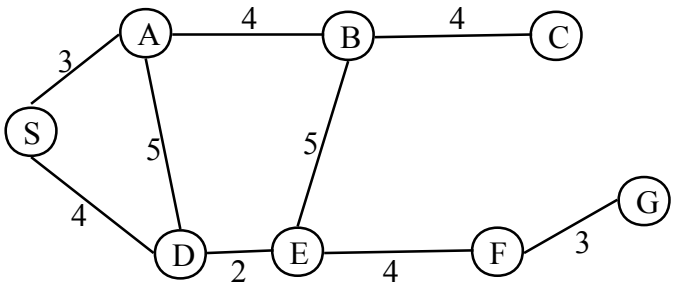


Figure : A graph representation of a map

This problem will be used to illustrate some search methods.

Search problems are part of a large number of real world applications:

- VLSI layout
- Path planning
- Robot navigation etc.

There are two broad classes of search methods:

- **uninformed (or blind) search methods;**
- **heuristically informed search methods.**

In the case of the uninformed search methods, the order in which potential solution paths are considered is arbitrary, using no domain-specific information to judge where the solution is likely to lie.

In the case of the heuristically informed search methods, one uses domain-dependent (heuristic) information in order to search the space more efficiently.

Measuring problem Solving Performance

We will evaluate the performance of a search algorithm in four ways

- **Completeness**
  An algorithm is said to be complete if it definitely finds solution to the problem, if exist.

- **Time Complexity**

How long (worst or average case) does it take to find a solution? Usually measured in terms of the **number of nodes expanded**

- **Space Complexity**
  How much space is used by the algorithm? Usually measured in terms of the **maximum number of nodes in memory at a time**

- **Optimality/Admissibility**
  If a solution is found, is it guaranteed to be an optimal one? For example, is it the one with minimum cost?

Time and space complexity are measured in terms of

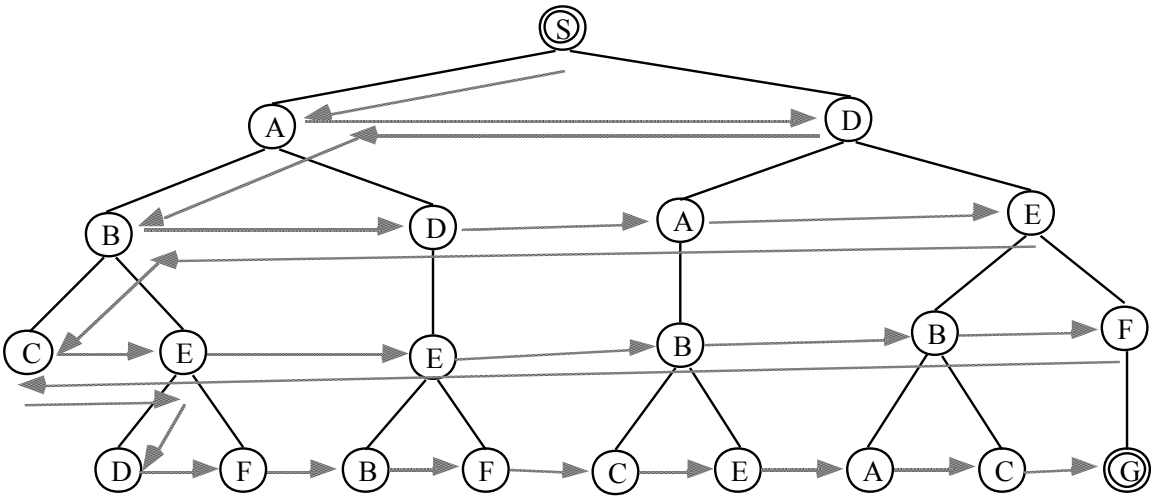
b -- maximum branching factor (number of successor of any node) of the search tree
d -- depth of the least-cost solution
m -- maximum length of any path in the space

Breadth First Search

All nodes are expended at a given depth in the search tree before any nodes at the next level are expanded until the goal reached.

Expand *shallowest* unexpanded node.

Constraint: Do not generate as child node if the node is already parent to avoid more loop.



BFS Evaluation:

Completeness:

- *Does it always find a solution if one exists?*
- YES
  - If shallowest goal node is at some finite depth d and If *b* is finite

Time complexity:

- Assume a state space where every state has *b* successors.
  - root has *b* successors, each node at the next level has again b successors (total b^2), …
  - Assume solution is at depth *d*
  - Worst case; expand all except the last node at depth *d*
  - Total no. of nodes generated:

$$b + b^2 + b^3 + \ldots\ldots\ldots\ldots\ldots\ldots b^d + (b^{d+1} - b) = \mathbf{O(b^{d+1})}$$

Space complexity:

- Each node that is generated must remain in memory
- Total no. of nodes in memory:

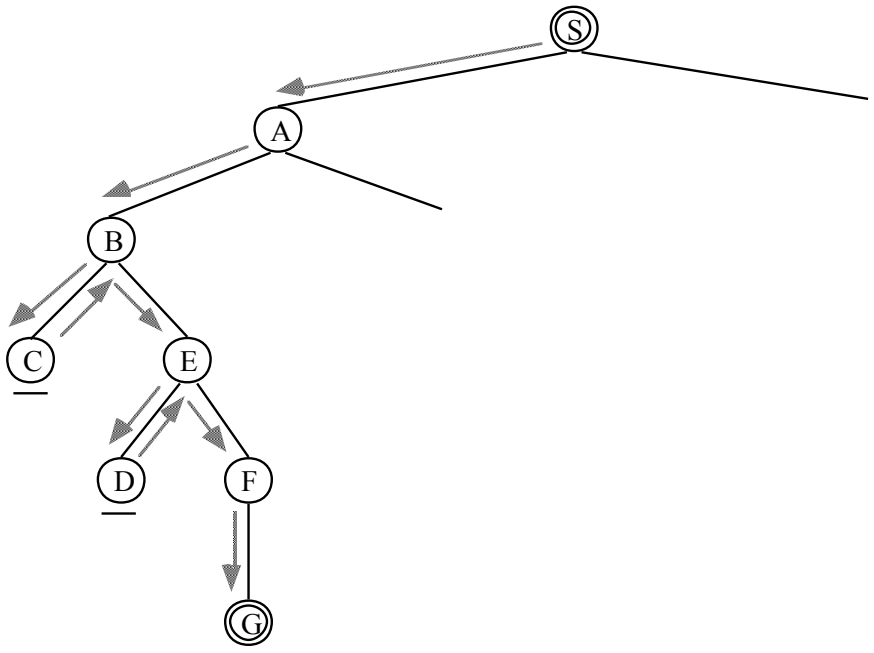
$$1 + b + b^2 + b^3 + \ldots\ldots\ldots\ldots\ldots\ldots b^d + (b^{d+1} - b) = \mathbf{O(b^{d+1})}$$

Optimal (i.e., admissible):

- if all paths have the same cost. Otherwise, not optimal but finds solution with shortest path length (shallowest solution). If each path does not have same path cost shallowest solution may not be optimal

Depth First Search

Looks for the goal node among all the children of the current node before using the sibling of this node i.e. **expand *deepest* unexpanded node.**



BFS Evaluation:

Completeness;

- *Does it always find a solution if one exists?*
- NO
  - If search space is infinite and search space contains loops then DFS may not find solution.

Time complexity;

- Let m is the maximum depth of the search tree. In the worst case Solution may exist at depth m.
- root has *b* successors, each node at the next level has again b successors (total b^2), …
- Worst case; expand all except the last node at depth *m*
- Total no. of nodes generated:

  $b + b^2 + b^3 + \ldots\ldots\ldots\ldots\ldots\ldots b^m = \mathbf{O(b^m)}$

Space complexity:

- It needs to store only a single path from the root node to a leaf node, along with remaining unexpanded sibling nodes for each node on the path.
- Total no. of nodes in memory:

  1+ b + b + b + …….. b m times = **O(bm)**

Optimal (i.e., admissible):

- DFS expand deepest node first, if expands entire let sub-tree even if right sub-tree contains goal nodes at levels 2 or 3. Thus we can say DFS may not always give optimal solution.

Heuristic Search:

Heuristic Search Uses domain-dependent (heuristic) information in order to search the space more efficiently.

Ways of using heuristic information:

- Deciding which node to expand next, instead of doing the expansion in a strictly breadth-first or depth-first order;
- In the course of expanding a node, deciding which successor or successors to generate, instead of blindly generating all possible successors at one time;
- Deciding that certain nodes should be discarded, or *pruned*, from the search space.

Informed Search uses domain specific information to improve the search pattern

- Define a heuristic function, h(n), that estimates the "goodness" of a node n.
- Specifically, h(n) = estimated cost (or distance) of minimal cost path from n to a goal state.
- The heuristic function is an estimate, based on domain-specific information that is computable from the current state description, of how close we are to a goal.

Best-First Search

Idea: use an *evaluation functionf(n)* that gives an indication of which node to expand next for each node.

- usually gives an estimate to the goal.
- the node with the lowest value is expanded first.

A key component of *f(n)* is a heuristic function, *h(n),*which is a additional knowledge of the problem.

There is a whole family of best-first search strategies, each with a different evaluation function.

Typically, strategies use estimates of the cost of reaching the goal and try to minimize it.

Special cases: based on the evaluation function.

- greedy best-first search
- A*search

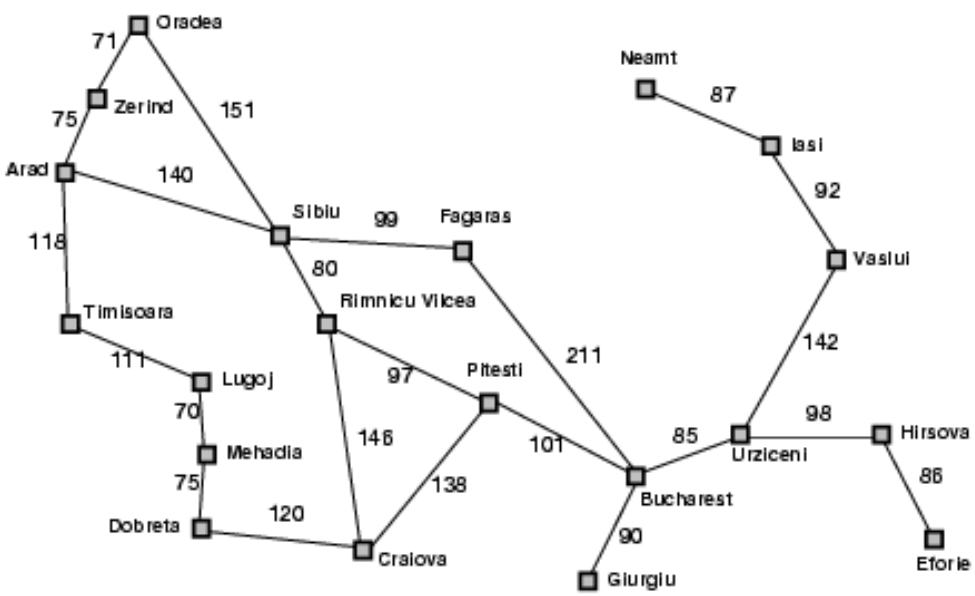
Greedy Best First Search

Evaluation function *f(n) = h(n)* (heuristic) = estimate of cost from *n* to *goal.*

e.g., $h_{SLD}(n)$ = straight-line distance from *n* to goal

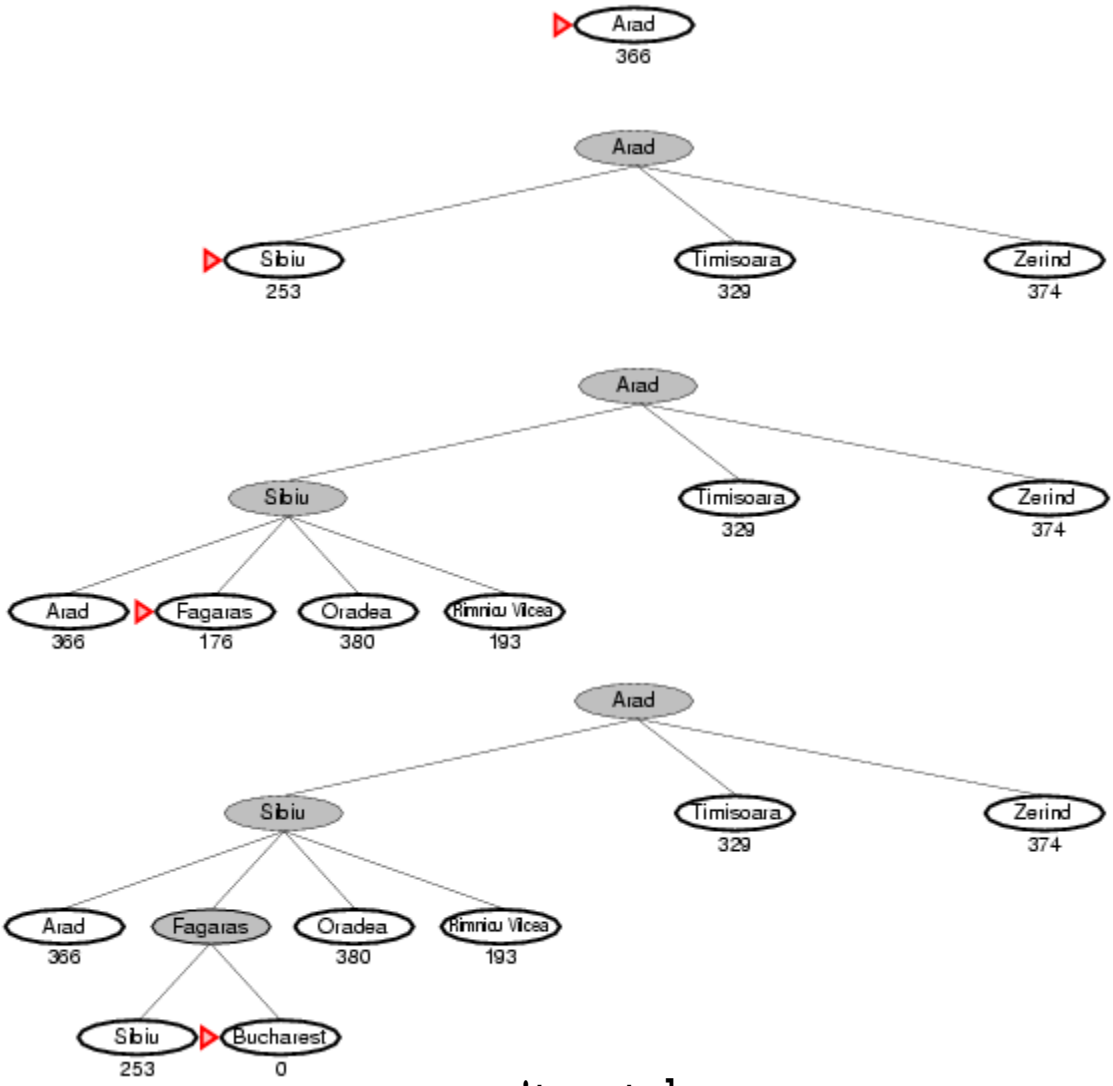
Greedy best-first search expands the node that appears to be closest to goal.

Example: Given following graph of cities, starting at Arad city, problem is to reach to the Bucharest.



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	176
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	10
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Solution using greedy best first can be as below:



A* Search : A Better Best-First Strategy

Greedy Best-first search

- minimizes estimated cost h(n) from current node n to goal;
- is informed but (almost always) suboptimal and incomplete.

Uniform cost search

- minimizes actual cost g(n) to current node n;
- is (in most cases) optimal and complete but uninformed.

A* search

- combines the two by minimizing f(n) = g(n) + h(n);
- is informed and, *under reasonable assumptions*, optimal and complete.

Idea: avoid expanding paths that are already expensive.

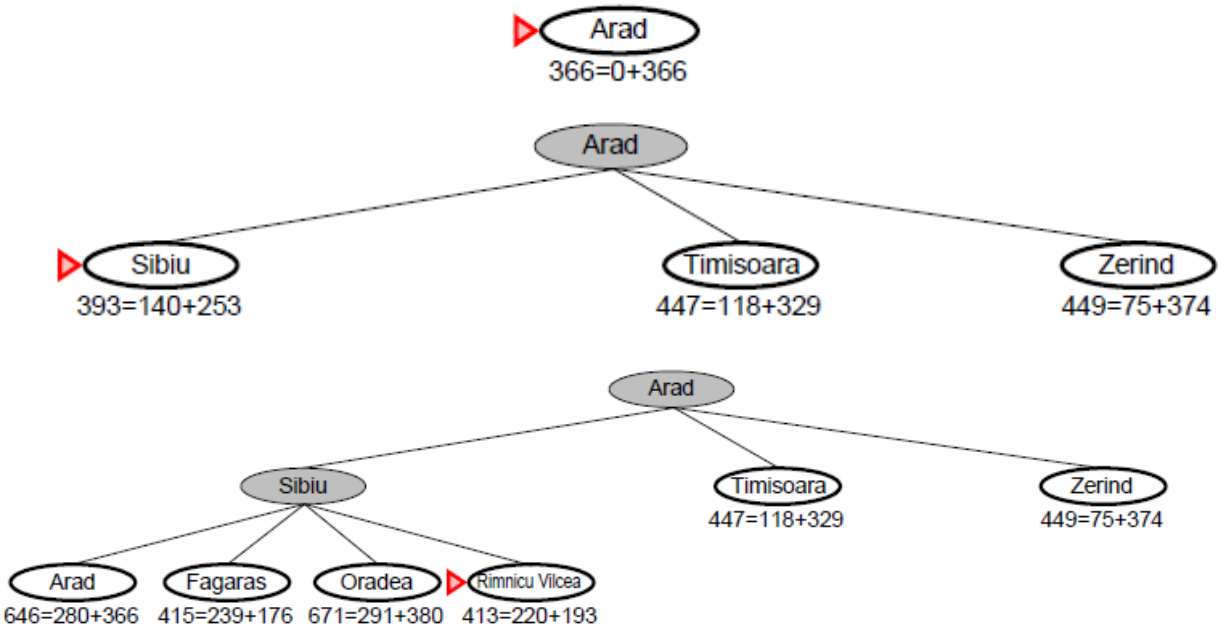
It finds a minimal cost-path joining the start node and a goal node for node n.

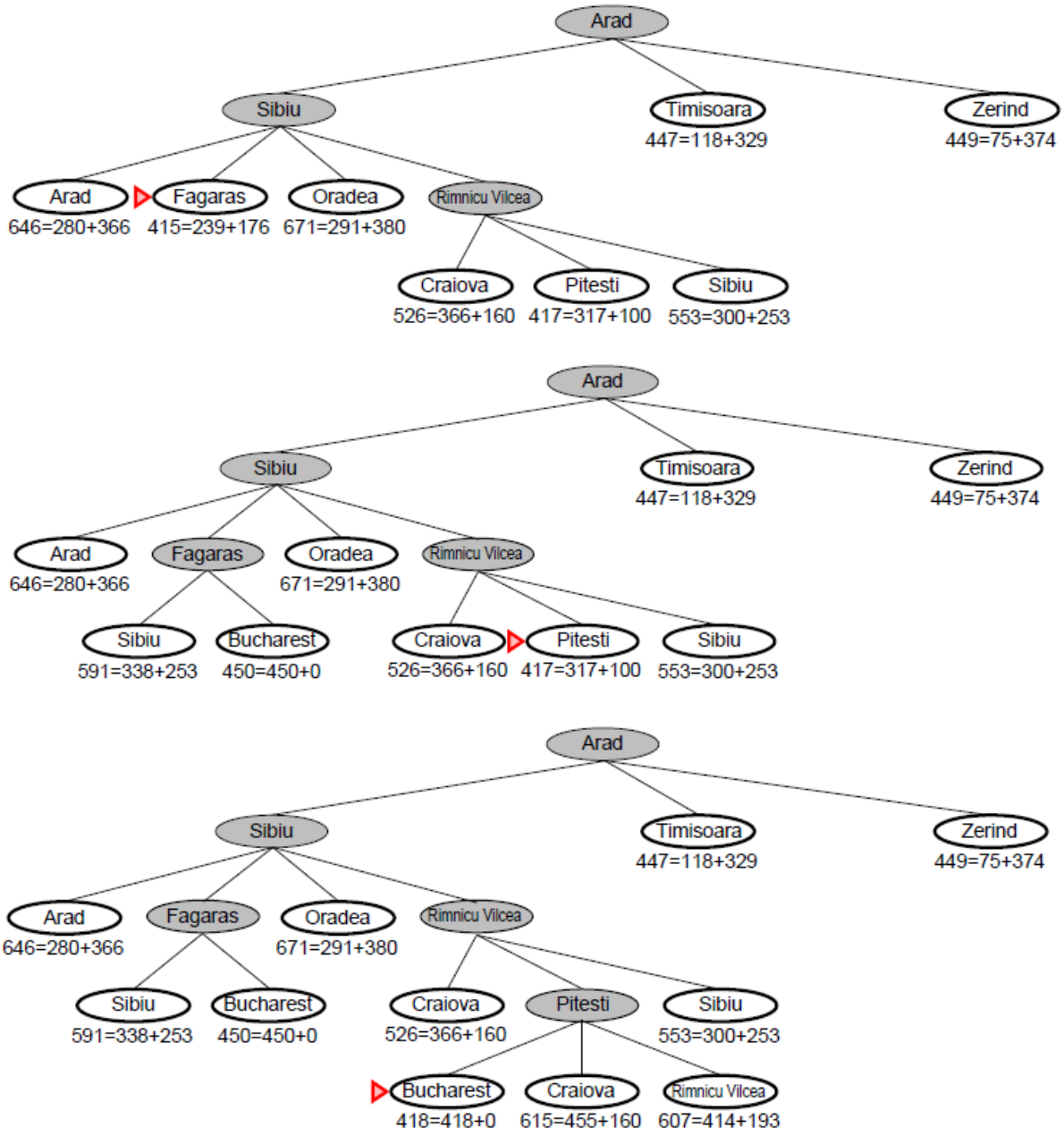
Evaluation function: f(n) = g(n) + h(n)

Where,

g(n) = cost so far to reach n from root
h(n) = estimated cost to goal from n
f(n) = estimated total cost of path through n to goal

A* Search Example:




Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
415=239+176
Oradea
671=291+380
Rimnicu Vilcea
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
417=317+100
Sibiu
553=300+253
Arad
Sibiu
Timisoara
447=118+329
Zerind
449=75+374
Arad
646=280+366
Fagaras
Oradea
671=291+380
Rimnicu Vilcea
Sibiu
591=338+253
Bucharest
450=450+0
Craiova
526=366+160
Pitesti
Sibiu
553=300+253
Bucharest
418=418+0
Craiova
615=455+160
Rimnicu Vilcea
607=414+193

Hill Climbing Search:

Hill climbing can be used to solve problems that have many solutions, some of which are better than others. **It starts with a random (potentially poor) solution, and iteratively makes small changes to the solution, each time improving it a little. When the algorithm cannot see any improvement anymore, it terminates.** Ideally, at that point the current solution is close to optimal, but it is not guaranteed that hill climbing will ever come close to the optimal solution.

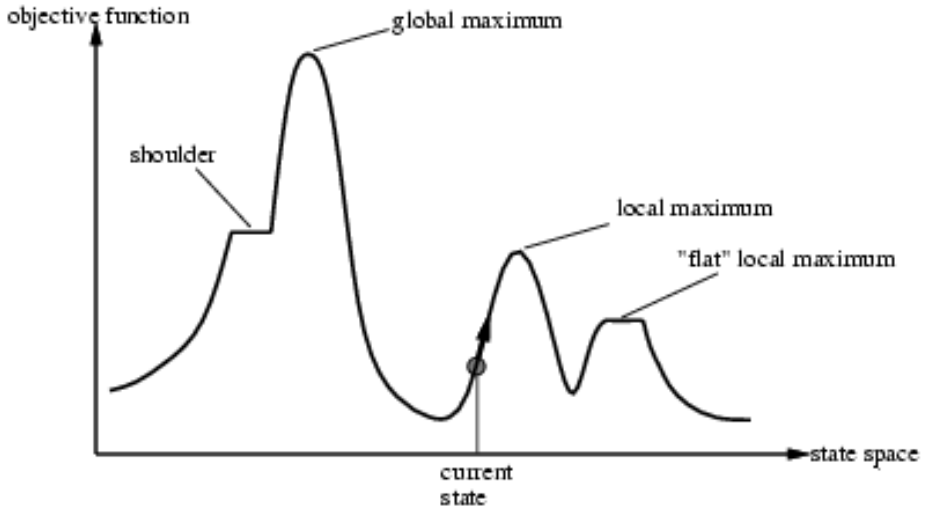
For example, hill climbing can be applied to the traveling salesman problem. It is easy to find a solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which two cities are visited. Eventually, a much better route is obtained.

In hill climbing the basic idea is to always head towards a state which is better than the current one. So, if you are at town A and you can get to town B and town C (and your target is town D) then you should make a move IF town B or C appear nearer to town D than town A does.

This can be described as follows:

1. Start with *current-state* = initial-state.
2. Until *current-state* = goal-state OR there is no change in *current-state* do:
    - Get the successors of the current state and use the evaluation function to assign a score to each successor.
    - If one of the successors has a better score than the current-state then set the new current-state to be the successor with the best score.

Hill climbing terminates when there are no successors of the current state which are better than the current state itself.



Expert Systems:

Definition:

An Expert system is a set of program that manipulates encoded knowledge to solve problem in a specialized domain that normally requires human expertise.

A computer system that simulates the **decision- making process** of a human expert in a specific domain.

An expert system's knowledge is obtained from expert sources and coded in a form suitable for the system to use in its inference or reasoning processes. The expert knowledge must be obtained from specialists or other sources of expertise, such as texts, journals, articles and data bases.

An expert system is an "intelligent" program that solves problems in a narrow problem area by using high-quality, specific knowledge rather than an algorithm.

Block Diagram

There is currently no such thing as "standard" expert system. Because a variety of techniques are used to create expert systems, they differ as widely as the programmers who develop them and the problems they are designed to solve. However, the principal components of most expert systems are **knowledge base, an inference engine, and a user interface,** as illustrated in the figure.

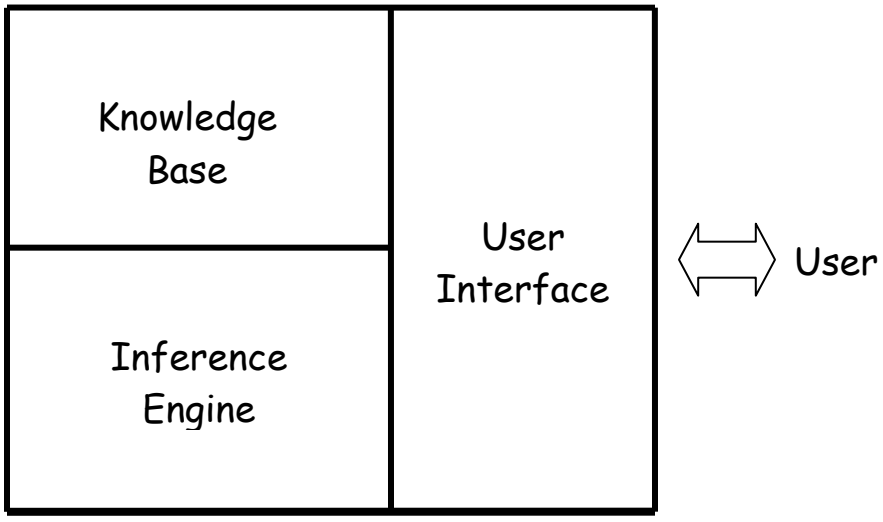


Fig: Block Diagram of expert system

1. Knowledge Base

The component of an expert system that contains the system's knowledge is called its knowledge base. This element of the system is so critical to the way most expert systems are constructed that they are also popularly known as *knowledge-based systems*

A knowledge base contains both declarative knowledge (facts about objects, events and situations) and procedural knowledge (information about courses of action). Depending on the form of knowledge representation chosen, the two types of knowledge may be separate or integrated. Although many knowledge representation techniques have been used in expert systems, the most prevalent form of knowledge representation currently used in expert systems is the *rule-based production* system approach.

To improve the performance of an expert system, we should supply the system with some knowledge about the knowledge it posses, or in other words, meta-knowledge.

2. Inference Engine

Simply having access to a great deal of knowledge does not make you an expert; you also must know **how** and **when** to apply the appropriate knowledge. Similarly, just having a knowledge base does not make an expert system intelligent. The system must have another component that directs the implementation of the knowledge. That element of the system is known variously as the *control structure*, the *rule interpreter*, or the *inference engine*.

The inference engine decides which heuristic search techniques are used to determine how the rules in the knowledge base are to be applied to the problem. In effect, an inference engine "runs" an expert system, determining which rules are to be invoked, accessing the appropriate rules in the knowledge base, executing the rules , and determining when an acceptable solution has been found.

3. User Interface

The component of an expert system that communicates with the user is known as the *user interface*. The communication performed by a user interface is bidirectional. At the simplest level, we must be able to describe our problem to the expert system, and the system must be able to respond with its recommendations. We may want to ask the system to explain its "reasoning", or the system may request additional information about the problem from us.

Beside these three components, there is a Working Memory - a data structure which stores information about a specific run. It holds current facts and knowledge.

Stages of Expert System Development:

Although great strides have been made in expediting the process of developing an expert system, it often remains an extremely time consuming task. It may be possible for one or two people to develop a small expert system in a few months; however the development of a sophisticated system may require a team of several people working together for more than a year.

An expert system typically is developed and refined over a period of several years. We can divide the process of expert system development into five distinct stages. In practice, it may not be possible to break down the expert system development cycle precisely. However, an examination of these five stages may serve to provide us with some insight into the ways in which expert systems are developed.

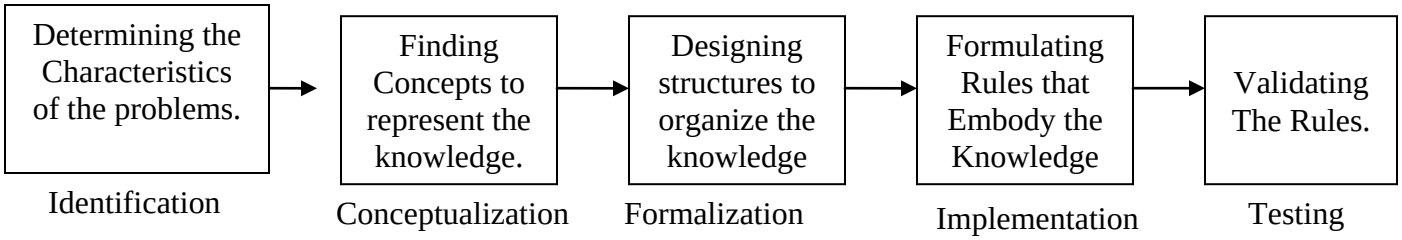


Fig: Different phases of expert system development

Identification:

Beside we can begin to develop an expert system, it is important that we describe, with as much precision as possible, the problem that the system is intended to solve. It is not enough simply to feel that the system would be helpful in certain situation; we must determine the exact nature of the problem and state the precise goals that indicate exactly how we expect the expert system to contribute to the solution.

Conceptualization:

Once we have formally identified the problem that an expert system is to solve, the next stage involves analyzing the problem further to ensure that its specifics, as well as it generalities, are understood. In the conceptualization stage the knowledge engineer frequently creates a diagram of the problem to depict graphically the relationships between the objects and processes in the problem domain. It is often helpful at this stage to divide the problem into a series of sub-problems and to diagram both the relationships among the pieces of each sub-problem and the relationships among the various sub-problems.

Formalization:

In the preceding stages, no effort has been made to relate the domain problem to the artificial intelligence technology that may solve it. During the identification and the conceptualization stages, the focus is entirely on understanding the problem. Now, during the formalization stage, the problem is connected to its proposed solution, an expert system, by analyzing the relationships depicted in the conceptualization stage.
During formalization, it is important that the knowledge engineer be familiar with the following:

- The various techniques of knowledge representation and heuristic search used in expert systems.
- The expert system "tools" that can greatly expedite the development process. And
- Other expert systems that may solve similar problems and thus may be adequate to the problem at hand.

Implementation:

During the implementation stage, the formalized concepts are programmed onto the computer that has been chosen for system development, using the predetermined techniques and tools to implement a "first pass" prototype of the expert system.

Theoretically, if the methods of the previous stage have been followed with diligence and care, the implementation of the prototype should be as much an art as it is a science, because following all rules does not guarantee that the system will work the first time it is implemented. Many scientists actually consider the first prototype to be a "throw-away' system, useful for evaluating progress but hardly a usable expert system.

Testing:

Testing provides opportunities to identify the weakness in the structure and implementation of the system and to make the appropriate corrections. Depending on the types of problems encountered, the testing procedure may indicate that the system was

Features of an expert system:

What are the features of a good expert system ? Although each expert system has its own particular characteristics, there are several features common to many systems. The following list from Rule-Based Expert Systems suggests seven criteria that are important prerequisites for the acceptance of an expert system .

1. "The program should be **useful**." An expert system should be developed to meet a specific need, one for which it is recognized that assistance is needed.

2. "The program should be **usable**." An expert system should be designed so that even a novice computer user finds it easy to use .

3. "The program should be **educational when appropriate**." An expert system may be used by non-experts, who should be able to increase their own expertise by using the system.

4. "The program should be able to **explain its advice**." An expert system should be able to explain the "reasoning" process that led it to its conclusions, to allow us to decide whether to accept the system's recommendations.

5. "The program should be able to **respond to simple questions**." Because people with different levels of knowledge may use the system , an expert system should be able to answer questions about points that may not be clear to all users.

6. "The program should be able to **learn new knowledge**." Not only should an expert system be able to respond to our questions, it also should be able to ask questions to gain additional information.

7. "The program's knowledge should be **easily modified**." It is important that we should be able to revise the knowledge base of an expert system easily to correct errors or add new information.

Neural Network

A neuron is a cell in brain whose principle function is the collection, Processing, and dissemination of electrical signals. Brains Information processing capacity comes from networks of such neurons. Due to this reason some earliest AI work aimed to create such artificial networks. (Other Names are Connectionism; Parallel distributed processing and neural computing).

What is a Neural Network?

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurones) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process.

Why use neural networks?

Neural networks, with their remarkable ability to derive meaning from complicated or imprecise data, can be used to extract patterns and detect trends that are too complex to be noticed by either humans or other computer techniques. A trained neural network can be thought of as an "expert" in the category of information it has been given to analyze. Other advantages include:

1. Adaptive learning: An ability to learn how to do tasks based on the data given for training or initial experience.
2. Self-Organisation: An ANN can create its own organisation or representation of the information it receives during learning time.
3. Real Time Operation: ANN computations may be carried out in parallel, and special hardware devices are being designed and manufactured which take advantage of this capability.
4. Fault Tolerance via Redundant Information Coding: Partial destruction of a network leads to the corresponding degradation of performance. However, some network capabilities may be retained even with major network damage

Neural networks versus conventional computers

Neural networks take a different approach to problem solving than that of conventional computers. Conventional computers use an algorithmic approach i.e. the computer follows a set of instructions in order to solve a problem. Unless the specific steps that the computer needs to follow are known the computer cannot solve the problem. That restricts the problem solving capability of conventional computers to problems that we already understand and know how to solve. But computers would be so much more useful if they could do things that we don't exactly know how to do.

Neural networks process information in a similar way the human brain does. The network is composed of a large number of highly interconnected processing elements(neurones) working in parallel to solve a specific problem. Neural networks learn by example. They cannot be programmed to perform a specific task. The examples must be selected carefully otherwise useful time is wasted or even worse the network might be functioning incorrectly. The disadvantage is that because the network finds out how to solve the problem by itself, its operation can be unpredictable.

On the other hand, conventional computers use a cognitive approach to problem solving; the way the problem is to solved must be known and stated in small unambiguous instructions. These instructions are then converted to a high level language program and then into machine code that the computer can understand. These machines are totally predictable; if anything goes wrong is due to a software or hardware fault.

Units of Neural Network

Nodes(units):

Nodes represent a cell of neural network.

Links:

Links are directed arrows that show propagation of information from one node to another node.

Activation:

Activations are inputs to or outputs from a unit.

Wight:

Each link has weight associated with it which determines strength and sign of the connection.

Activation function:

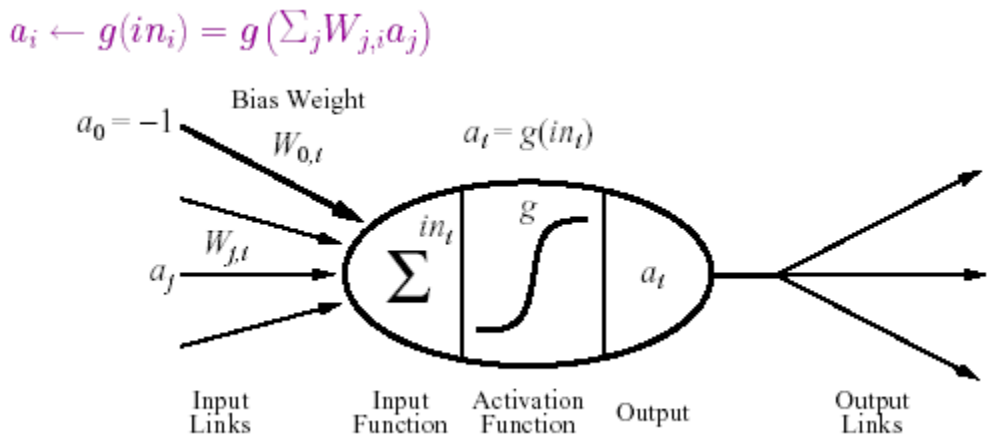
A function which is used to derive output activation from the input activations to a given node is called activation function.

Bias Weight:

Bias weight is used to set the threshold for a unit. Unit is activated when the weighted sum of real inputs exceeds the bias weight.

Simple Model of Neural Network

A simple mathematical model of neuron is devised by McCulloch and Pit is given in the figure given below:



It fires when a linear combination of its inputs exceeds some threshold.

A neural network is composed of nodes (units) connected by directed links A link from unit j to i serve to propagate the activation a_j from j to i. Each link has some numeric weight $W_{j,i}$ associated with it, which determines strength and sign of connection.

Each unit first computes a weighted sum of it's inputs:

$$in_i = \sum_{J=0}^{n} W_{j,i}\ a_j$$

Then it applies activation function g to this sum to derive the output:

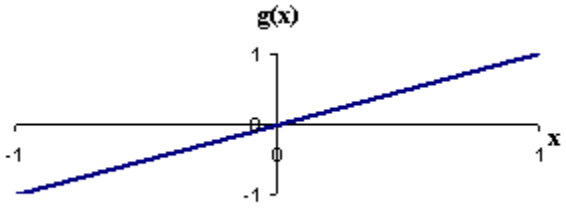
$$a_i = g(\ in_i) = g(\sum_{J=0}^{n} W_{j,i}\ a_j)$$

Here, a_j output activation from unit j and $W_{j,i}$ is the weight on the link j to this node. Activation function typically falls into one of three categories:

- Linear
- Threshold (*Heaviside function)*
- Sigmoid
- Sign

For **linear activation functions**, the output activity is proportional to the total weighted output.

g(x) = k x + c, where k and x are constant



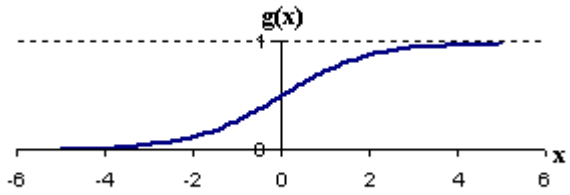
For **threshold activation functions**, the output are set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

g(x) = 1 if x>= k

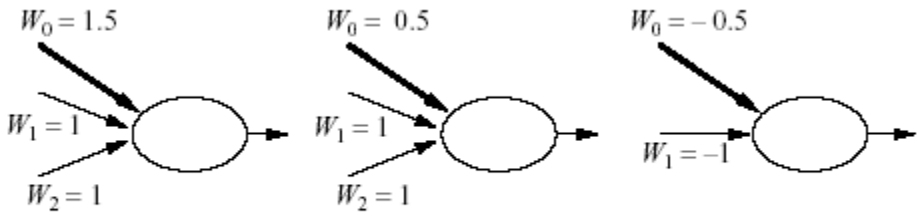
= 0 if x < k

For **sigmoid activation functions**, the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units. It has the advantage of differentiable.

$g(x) = 1/ (1 + e^{-x})$



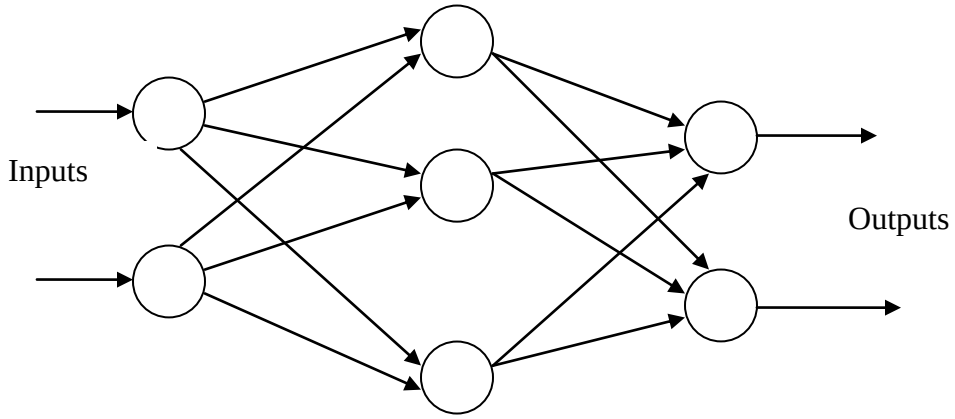
Realizing logic gates by using Neurons:



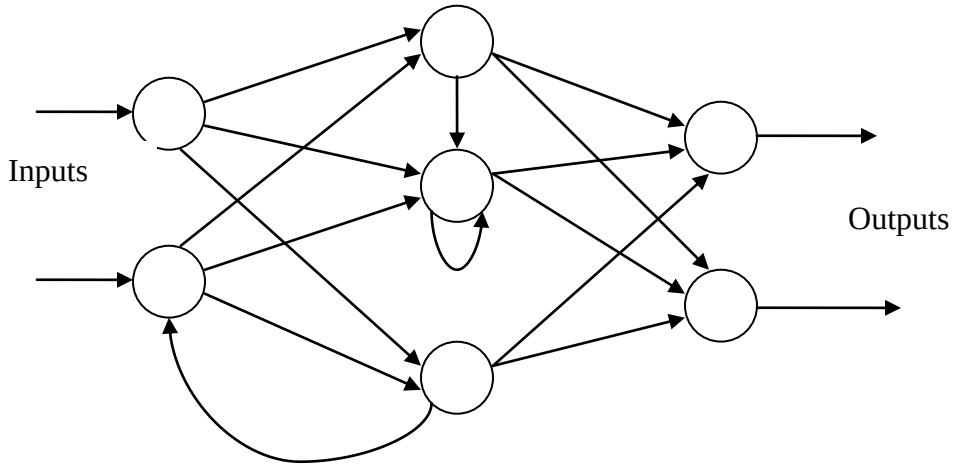
Network structures

Feed-forward networks:

Feed-forward ANNs allow signals to travel one way only; from input to output. There is no feedback (loops) i.e. the output of any layer does not affect that same layer. Feed-forward ANNs tend to be straight forward networks that associate inputs with outputs. They are extensively used in pattern recognition. This type of organization is also referred to as bottom-up or top-down.

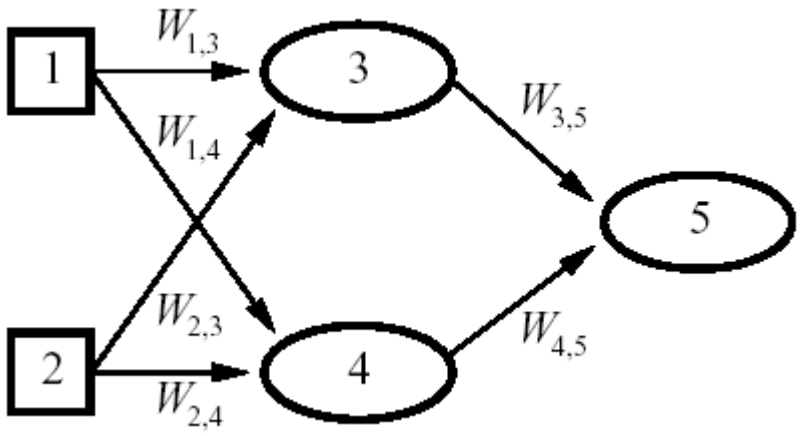


Feedback networks (Recurrent networks:)



Feedback networks (figure 1) can have signals traveling in both directions by introducing loops in the network. Feedback networks are very powerful and can get extremely complicated. Feedback networks are dynamic; their 'state' is changing continuously until they reach an equilibrium point. They remain at the equilibrium point until the input changes and a new equilibrium needs to be found. Feedback architectures are also referred to as interactive or recurrent.

Feed-forward example



Here;

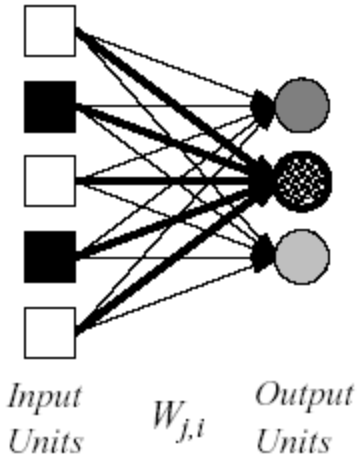
$a5 = g(W_{3;5}\ a_3 + W_{4;5}\ a_4)$

$= g(W_{3;5}\ g(W_{1;3}\ a_1 + W_{2;3}\ a_2) + W4;5\ g(W_{1;4}\ a_1 + W_{2;4}\ a_2)$

Types of Feed Forward Neural Network:

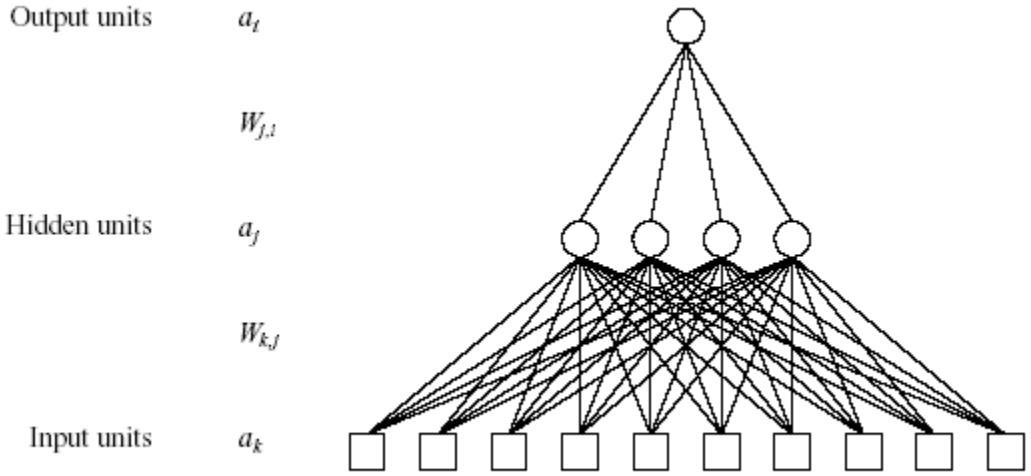
Single-layer neural networks (perceptrons)

A neural network in which all the inputs connected directly to the outputs is called a single-layer neural network, or a perceptron network. Since each output unit is independent of the others each weight affects only one of the outputs.



Multilayer neural networks (perceptrons)

The neural network which contains input layers, output layers and some hidden layers also is called multilayer neural network. The advantage of adding hidden layers is that it enlarges the space of hypothesis. Layers of the network are normally fully connected.



Training in Neural Network

Once the number of layers, and number of units in each layer, has been selected, training is used to set the network's weights and thresholds so as to minimize the prediction error made by the network

Training is the process of adjusting weights and threshold to produce the desired result for different set of data.

[Unit -3: Computation]

Introduction to Cognitive Science (CSC-255)

Central Department of Computer Science & Information Technology
Tribhuvan University

Unit 3: Computation

Computation: If it involves a computer, a program running on a computer and numbers going in and out then **computation** is likely happening.

The notion that mental states and processes intervene between stimuli and responses sometimes takes the form of a "computational" metaphor or analogy, which is often used as the identifying mark of contemporary cognitive science: The mind is to the brain as software is to hardware; mental states and processes are (like) computer programs implemented (in the case of humans) in brain states and processes. Some cognitive scientists make the stronger claim that mental states and processes *are* (expressible as) algorithms: "cognition *is* a type of computation" (Pylyshyn 1985: xiii). Others make a weaker, but more general, claim that cognition is comput*able*, i.e., that there are algorithms that have the same input–output behavior as cognitive processes (cf. Rapaport, forthcoming).

Thus, according to the computational view of cognitive science, (1) there are mental states and processes intervening between input stimuli and output responses, (2) these mental states and processes either *are* computations or else are comput*able*, and—hence—(3) in contrast to behaviorism, mental states and processes are capable of being investigated scientifically (even if they are not capable of being directly observed).

Insofar as the methods of investigation are taken to be computational in nature, computer science in general and artificial intelligence in particular have come to play a central role in cognitive science. It is, however, a role not without controversial philosophical implications: For if mental states and processes can be expressed as algorithms, then they are capable of being implemented in non-human computers. The philosophical issue is simply this: Are computers executing such algorithms merely simulating mental states and processes, or are they actually exhibiting them? Do such computers think?

In computability theory and computational complexity theory, a **model of computation** is the definition of the set of allowable operations used in computation and their respective costs. Only assuming a certain model of computation is it possible to analyze the computational resources required, such as the execution time or memory space or to discuss the limitations of algorithms or computers.

Some examples of models include Turing machines, recursive functions, lambda calculus, and production systems.

Models of computation:

Artificial neural networks can be considered as just another approach to the problem of computation. The first formal definitions of computability were proposed in the 1930s and '40s and at least five different alternatives were studied at the time. The computer era was started, not with one single approach, but with a contest of alternative computing models. We all know that the von Neumann computer emerged as the undisputed winner in this confrontation, but its triumph did not lead to the dismissal of the other computing models. Figure 1.1 shows the five principal contenders:

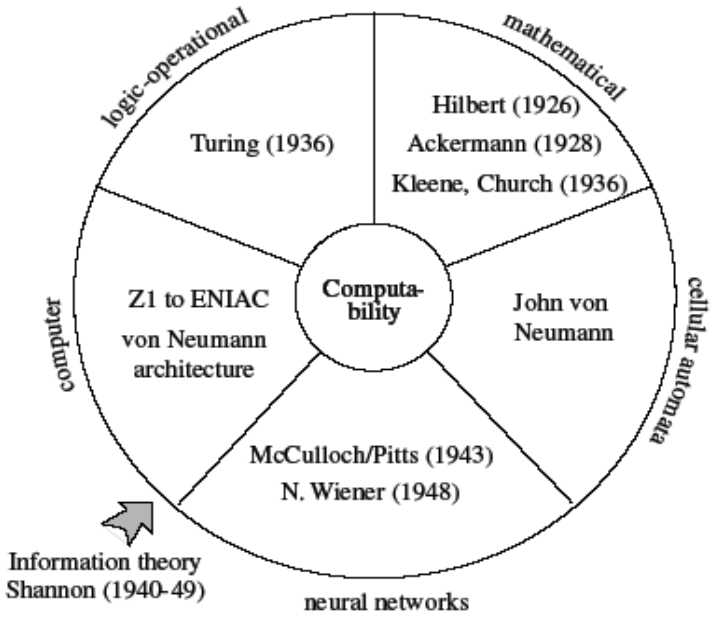


Fig. 1.1. Five models of computation

- The Mathematical Model:

David Hilbert, the famous German mathematician, was the first to state the **conjecture** that a **certain class of functions contains all intuitively computable functions**. Hilbert was referring to the primitive recursive functions, the class of functions which can be constructed from the zero and successor function using composition, projection, and a deterministic number of iterations (primitive recursion). However, in 1928, **Wilhelm Ackermann** was able to find **a computable function** which is not primitive recursive. This led to the definition of the general recursive functions. In this formalism, a new composition rule has to be introduced, the so-called μ operator, which is equivalent to an indeterminate recursion or a lookup in an infinite table. At the same time **Alonzo Church** and collaborators developed the **lambda calculus**, another alternative to the mathematical definition of the computability concept. In 1936, **Church and Kleene** were able to show that the general recursive functions can be expressed in the formalism of the lambda calculus. This led to the Church thesis that computable functions are the general recursive functions.

- The Logic-Operational Model (Turing Machine):

Alan Turing introduced another kind of computing model. The advantage of his approach is that it consists in an operational, mechanical model of computability. A Turing machine is composed of an **infinite tape**, in which symbols can be stored and read again. A read-write head can move to the left or to the right according to its internal state, which is updated at each step. The Turing thesis states that computable functions are those which can be computed with this kind of device. It was formulated concurrently with the Church thesis and Turing was able to show almost immediately that they are equivalent. The Turing approach

made clear for the first time what "programming" means, curiously enough at a time when no computer had yet been built.

- The Computer Model:

The first electronic computing devices were developed in the 1930s and '40s. Since then, "**computation-with-the-computer**" has been regarded as computability itself. Computers of the time, like the **Mark I** built at Harvard, could iterate a constant number of times but were incapable of executing open-ended iterations (WHILE loops). Therefore the Mark I could compute the primitive but not the general recursive functions. Also the **ENIAC**, which is usually hailed as the world's first electronic computer, was incapable of dealing with open-ended loops, since iterations were determined by specific connections between modules of the machine. It seems that the first universal computer was the Mark I built in Manchester. This machine was able to cover all computable functions by making use of conditional branching and self-modifying programs.

- Cellular Automata:

The history of the development of the first mechanical and electronic computing devices shows how difficult it was to reach a consensus on the architecture of universal computers. Aspects such as the economy or the dependability of the building blocks played a role in the discussion, but the main problem was the definition of the minimal architecture needed for universality. **In machines like the Mark I and the ENIAC there was no clear separation between memory and processor, and both functional elements were intertwined. Some machines still worked with base 10 and not 2, some were sequential and others parallel**. John von Neumann, who played a major role in defining the architecture of sequential machines, analyzed at that time a new computational model which he called **cellular automata**. **Such automata operate in a "computing space" in which all data can be processed simultaneously. The main problem for cellular automata is communication and coordination between all the computing** cells. This can be guaranteed through certain algorithms and conventions. It is not difficult to show that all computable functions, in the sense of Turing, can also be computed with cellular automata, even of the one-dimensional type, possessing only a few states. Turing himself considered this kind of computing model at one point in his career.

- The biological model (neural networks):

The explanation of important aspects of the physiology of neurons set the stage for the formulation of **artificial neural network models which do not operate sequentially, as Turing machines do**. Neural networks have a hierarchical multilayered structure which sets them apart from cellular automata, so that information is transmitted not only to the immediate neighbors but also to more distant units. In artificial neural networks one can connect each unit to any other. In contrast to conventional computers, no program is handed over to the hardware – such a program has to be created, that is, the free parameters of the network have to be found adaptively.

Elements of a computing model:

What are the elementary components of any conceivable computing model? In the theory of general recursive functions, for example, it is possible to reduce any computable function to some composition rules and a small set of primitive functions. For a universal computer, we ask about the existence of a minimal and sufficient instruction set. For an arbitrary computing model the following metaphoric expression has been proposed:

Computation = storage + transmission + processing.

The mechanical computation of a function presupposes that these three elements are present, that is, that data can be stored, communicated to the functional units of the model and transformed. It is implicitly assumed that a certain coding of the data has been agreed upon.

Modern computers transform storage of information into a form of information transmission. Static memory chips store a bit as a circulating current until the bit is read. Turing machines store information in an infinite tape, whereas transmission is performed by the read-write head. Cellular automata store information in each cell, which at the same time is a small processor.

Self Study: DFA, NFA, and PDA (with practical examples)

The Turing Machine:

A **Turing machine** is a theoretical device that manipulates symbols contained on a strip of tape. Despite its simplicity, a Turing machine can be adapted to simulate the logic of any computer algorithm, and is particularly useful in explaining the functions of a CPU inside of a computer. The "Turing" machine was described by **Alan Turing** in 1937, who called it an "automatic-machine". Turing machines are not intended as a practical computing technology, but rather as a **thought experiment** representing a computing machine. They help computer scientists understand the limits of mechanical computation.

The Turing machine mathematically models a machine that mechanically operates on a tape on which symbols are written which it can read and write one at a time using a tape head. Operation is fully determined by a finite set of elementary instructions such as "in state 2, if the symbol seen is 0, write a 1; if the symbol seen is 1, shift to the right, and change into state 3; in state 3, if the symbol seen is 0, write a 1 and change to state 4;" etc.

More precisely, a Turing machine consists of:

1. A **TAPE** which is divided into cells, one next to the other. Each cell contains a symbol from some finite alphabet. The alphabet contains a special *blank* symbol (here written as '0') and one or more other symbols. The tape is assumed to be arbitrarily extendable to the left and to the right, i.e., the Turing machine is always supplied with as much tape as it needs for its computation. Cells that have not been written to before are assumed to be filled with the blank symbol. In some models the tape has a

left end marked with a special symbol; the tape extends or is indefinitely extensible to the right.

2. A **HEAD** that can read and write symbols on the tape and move the tape left and right one (and only one) cell at a time. In some models the head moves and the tape is stationary.

3. A finite **TABLE** ("action table", or *transition function*) of instructions : $q_i a_j \rightarrow q_{i1} a_{j1} d_k$, that, given the *state*(q_i) the machine is currently in *and* the *symbol*(a_j) it is reading on the tape (symbol currently under HEAD) tells the machine to do the following in sequence:
    - Either erase or write a symbol (instead of a_j written a_{j1}), *and then*
    - Move the head (which is described by d_k and can have values: 'L' for one step left *or* 'R' for one step right *or* 'S' for staying in the same place), *and then*
    - Assume the same or a *new state* as prescribed (go to state q_{i1}).

4. A **STATE REGISTER** that stores the state of the Turing table, one of finitely many. There is one special *start state* with which the state register is initialized. These states, writes Turing, replace the "state of mind" a person performing computations would ordinarily be in.

Note that every part of the machine—its state and symbol-collections—and its actions—printing, erasing and tape motion—is finite, discrete and distinguishable; it is the potentially unlimited amount of tape that gives it an unbounded amount of storage space.

Formal Definition:

Formally, we can define a Turing machine as a 7-tuple $M = \langle Q, \Gamma, b, \Sigma, \delta, q_0, F \rangle$ where

- Q is a finite set of *states*
- Γ is a finite set of the *tape alphabet/symbols*
- $b \in \Gamma$ is the *blank symbol* (the only symbol allowed to occur on the tape infinitely often at any step during the computation)
- $\Sigma \subseteq \Gamma \setminus \{b\}$ is the set of *input symbols*
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ is a function called the ***transition function***, where L is left shift, R is right shift. (A relatively uncommon variant allows "no shift", say N / S as stationary, as a third element of the latter set.)
- $q_0 \in Q$ is the *initial state*
- $F \subseteq Q$ is the set of *final* or *accepting states*.

Anything that operates according to these specifications is a Turing machine.

Illustration of TM can be done either through **transition table** or **transition diagram**.

Comparison of TM with real machines:

It is often said that Turing machines, unlike simpler automata, are as powerful as real machines, and are able to execute any operation that a real program can. What is missed in this statement is that, because a real machine can only be in finitely many *configurations*, in fact this "real machine" is nothing but a **deterministic finite automaton**. On the other hand, Turing machines are equivalent to machines that have an unlimited amount of storage space for their computations. In fact, Turing machines are not intended to model computers, but rather they are intended to model computation itself; historically, computers, which compute only on their (fixed) internal storage, were developed only later.

There are a number of ways to explain why Turing machines are useful models of real computers:

1. Anything a real computer can compute, a Turing machine can also compute. For example: "A Turing machine can simulate any type of subroutine found in programming languages, including recursive procedures and any of the known parameter-passing mechanisms". A large enough FSA can also model any real computer, disregarding IO. Thus, a statement about the limitations of Turing machines will also apply to real computers.

2. The difference lies only with the ability of a Turing machine to manipulate an unbounded amount of data. However, given a finite amount of time, a Turing machine (like a real machine) can only manipulate a finite amount of data.

3. Like a Turing machine, a real machine can have its storage space enlarged as needed, by acquiring more disks or other storage media. If the supply of these runs short, the Turing machine may become less useful as a model. But the fact is that neither Turing machines nor real machines need astronomical amounts of storage space in order to perform useful computation. **The processing time required is usually much more of a problem.**

4. Descriptions of real machine programs using simpler abstract models are often much more complex than descriptions using Turing machines. For example, a Turing machine describing an algorithm may have a few hundred states, while the equivalent **deterministic finite automaton** on a given real machine has quadrillions. This makes the DFA representation infeasible to analyze.

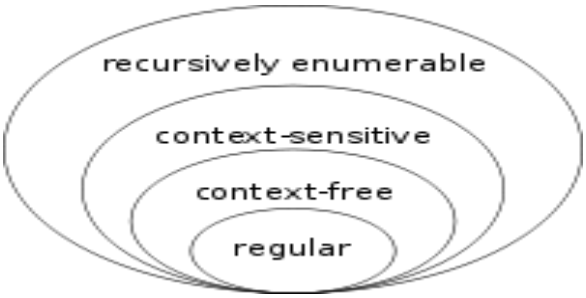
5. Turing machines describe algorithms independent of how much memory they use. There is a limit to the memory possessed by any current machine, but this limit can rise arbitrarily in time. Turing machines allow us to make statements about algorithms which will (theoretically) hold forever, regardless of advances in *conventional* computing machine architecture.

6. Turing machines simplify the statement of algorithms. Algorithms running on Turing-equivalent abstract machines are usually more general than their counterparts running on real machines, because they have arbitrary-precision data types available and never have

to deal with unexpected conditions (including, but not limited to, running out of memory).

Chomsky Hierarchy:

When **Noam Chomsky** first formalized generative grammars in 1956, he classified them into types now known as the Chomsky hierarchy. The difference between these types is that they have increasingly strict production rules and can express fewer formal languages. The Chomsky hierarchy consists of the following levels:



- **Type-0 grammars (unrestricted grammars)** include all formal grammars. They generate exactly all languages that can be recognized by a Turing machine. These languages are also known as the recursively enumerable languages. Note that this is different from the recursive languages which can be *decided* by an always-halting Turing machine.

- **Type-1 grammars (context-sensitive grammars)** generate the context-sensitive languages. These grammars have rules of the form $\alpha A\beta \rightarrow \alpha\gamma\beta$ with A a nonterminal and α, β and γ strings of terminals and nonterminals. The strings α and β may be empty, but γ must be nonempty. The rule $S \rightarrow \epsilon$ is allowed if S does not appear on the right side of any rule. The languages described by these grammars are exactly all languages that can be recognized by a linear bounded automaton (a nondeterministic Turing machine whose tape is bounded by a constant times the length of the input.)

- **Type-2 grammars (context-free grammars)** generate the context-free languages. These are defined by rules of the form $A \rightarrow \gamma$ with A a nonterminal and γ a string of terminals and nonterminals. These languages are exactly all languages that can be recognized by a non-deterministic pushdown automaton. Context-free languages are the theoretical basis for the syntax of most programming languages.

- **Type-3 grammars (regular grammars)** generate the regular languages. Such a grammar restricts its rules to a single nonterminal on the left-hand side and a right-hand side consisting of a single terminal, possibly followed (or preceded, but not both in the same grammar) by a single nonterminal. The rule $S \rightarrow \epsilon$ is also allowed here if S does not appear on the right side of any rule. These languages are exactly all languages that can be decided by a finite state automaton. Additionally, this family of

formal languages can be obtained by regular expressions. Regular languages are commonly used to define search patterns and the lexical structure of programming languages.

Unrestricted grammar (Type 0):

In formal language theory, an **unrestricted grammar** is a formal grammar on which no restrictions are made on the left and right sides of the grammar's productions.

An **unrestricted grammar** is a formal grammar $G = (N, \Sigma, P, S)$, where N is a set of nonterminal symbols, Σ is a set of terminal symbols, N and Σ are disjoint (actually, this is not strictly necessary, because unrestricted grammars make no real distinction between nonterminal and terminal symbols, the designation exists purely so that one knows when to stop when trying to generate sentential forms of the grammar), P is a set of production rules of the form $\alpha \rightarrow \beta$ where α and β are strings of symbols in $N \cup \Sigma$ and α is not the empty string, and $S \in N$ is a specially designated start symbol. As the name implies, there are no real restrictions on the types of production rules that unrestricted grammars can have.

Example. The following grammar is unrestricted. $S \rightarrow TbC$
$Tb \rightarrow \mathfrak{c}$
$cC \rightarrow Sc \mid \Lambda.$
This grammar is not context-sensitive, not context-free, and not regular.

In unrestricted grammar, the left hand side of a rule contains a string of terminals and non-terminals (at least one of which must be a non-terminal).

Unrestricted grammars and Turing machines

It may be shown that unrestricted grammars characterize the recursively enumerable languages. This is the same as saying that for every unrestricted grammar G there exists some Turing machine capable of recognizing $L(G)$ and vice-versa. Given an unrestricted grammar, such a Turing machine is simple enough to construct, as a two-tape nondeterministic Turing machine. The first tape contains the input word w to be tested, and the second tape is used by the machine to generate sentential forms from G. The Turing machine then does the following:

1. Start at the left of the second tape and repeatedly choose to move right or select the current position on the tape.
2. Non deterministically choose a production $\beta \rightarrow \gamma$ from the productions in G.
3. If β appears at some position on the second tape, replace β by γ at that point, possibly shifting the symbols on the tape left or right depending on the relative lengths of β and γ (e.g. if β is longer than γ, shift the tape symbols left).
4. Compare the resulting sentential form on tape 2 to the word on tape 1. If they match, then the Turing machine accepts the word. If they don't go back to step 1.

It is easy to see that this Turing machine will generate all and only the sentential forms of G on its second tape after the last step is executed an arbitrary number of times, thus the language $L(G)$ must be recursively enumerable.

The reverse construction is also possible. Given some Turing machine, it is possible to create an unrestricted grammar.

Context Sensitive Grammar (Type 1):

A **context-sensitive grammar** is a formal grammar in which the left-hand sides and right-hand sides of any production rules may be surrounded by a context of terminal and nonterminal symbols. Context-sensitive grammars are more general than context-free grammars but still orderly enough to be parsed by a linear bounded automaton.

A formal grammar $G = (N, \Sigma, P, S)$ is context-sensitive if all rules in P are of the form

$$\alpha A \beta \rightarrow \alpha \gamma \beta$$

where $A \in N$ (i.e., A is a single non terminal), $\alpha, \beta \in (N \cup \Sigma)^*$ (i.e., α and β are strings of nonterminals and terminals) and $\gamma \in (N \cup \Sigma)^+$ (i.e., γ is a nonempty string of nonterminals and terminals).

Some definitions also add that for any production rule of the form $u \rightarrow v$ of a context-sensitive grammar, it shall be true that $|u| \leq |v|$. Here $|u|$ and $|v|$ denote the length of the strings respectively.

In addition, a rule of the form

$S \rightarrow \lambda$ provided S does not appear on the right side of any rule

where λ represents the empty string is permitted. The addition of the empty string allows the statement that the context sensitive languages are a proper superset of the context free languages, rather than having to make the weaker statement that all context free grammars with no $\rightarrow\lambda$ productions are also context sensitive grammars.

The name *context-sensitive* is explained by the α and β that form the context of A and determine whether A can be replaced with γ or not. This is different from a context-free grammar where the context of a nonterminal is not taken into consideration.

If the possibility of adding the empty string to a language is added to the strings recognized by the noncontracting grammars (which can never include the empty string) then the languages in these two definitions are identical.

Example. The following grammar is context-sensitive.

$S \to aTb \mid ab$
$aT \to aaTb \mid ac.$

Quiz. What is the language of the grammar?

Answer: $\{ab\} \cup \{a^{n+1}cb^{n+1} \mid n \in \mathbf{N}\}$. This language is context-free. For example, it has the grammar $S \to aTb \mid ab$, and $T \to aTb \mid c$.

Any context-free language is context sensitive.

Example. $\{a^nb^nc^n \mid n \geq 0\}$ is context-sensitive but not context-free. Here is a csg.

$S \to \Lambda \mid abc \mid aTBc$
$T \to abC \mid aTBC$
$CB \to CX \to BX \to BC$
$bB \to bb.$
$Cc \to cc.$

Quiz. Derive *aaabbbccc*.

Answer: $S \Rightarrow aTBc \Rightarrow aaTBCBc \Rightarrow aaabCBCBc \Rightarrow aaabBCCBc \Rightarrow aaabBCBCc$
$\Rightarrow aaabBBCCc \Rightarrow aaabbBCCc \Rightarrow aaabbbCCc \Rightarrow aaabbbCcc \Rightarrow aaabbbccc.$

Similarly the grammar for $\{a^nb^nc^n : n \geq 1\}$ can be written as:

1. $S \to aSBC$
2. $S \to aBC$
3. $CB \to HB$
4. $HB \to HC$
5. $HC \to BC$
6. $aB \to ab$
7. $bB \to bb$
8. $bC \to bc$
9. $cC \to cc$

The generation chain for aaa bbb ccc is:

S
$\Rightarrow_1 aSBC$
$\Rightarrow_1 a\boldsymbol{aSBC}BC$
$\Rightarrow_2 aa\boldsymbol{aBC}BCBC$
$\Rightarrow_3 aaaB\boldsymbol{HB}CBC$
$\Rightarrow_4 aaaB\boldsymbol{HC}CBC$
$\Rightarrow_5 aaaB\boldsymbol{BC}CBC$
$\Rightarrow_3 aaaBBC\boldsymbol{HB}C$
$\Rightarrow_4 aaaBBC\boldsymbol{HC}C$
$\Rightarrow_5 aaaBBC\boldsymbol{BC}C$
$\Rightarrow_3 aaaBB\boldsymbol{HB}CC$
$\Rightarrow_4 aaaBB\boldsymbol{HC}CC$

$$\Rightarrow_5 aaaBB\boldsymbol{BC}CC$$
$$\Rightarrow_6 aa\boldsymbol{ab}BBCCC$$
$$\Rightarrow_7 aaa\boldsymbol{bb}BCCC$$
$$\Rightarrow_7 aaab\boldsymbol{bb}CCC$$
$$\Rightarrow_8 aaabb\boldsymbol{bc}CC$$
$$\Rightarrow_9 aaabbb\boldsymbol{cc}C$$
$$\Rightarrow_9 aaabbbc\boldsymbol{cc}$$

Context Free Grammar (Type 2):

In formal language theory, a **context-free grammar** (**CFG**), sometimes also called a **phrase structure grammar** is a grammar which naturally generates a formal language in which clauses can be nested inside clauses arbitrarily deeply, but where grammatical structures are not allowed to overlap.

The canonical example is matching parentheses: parentheses of different types must open and close correctly inside each other, like this:

([[[ ]() [ ][ ]]]([ ])])

like any context free grammars, the logical units, the contents of corresponding matched parentheses, nest cleanly.

In terms of production rules, every production of a context free grammar is of the form

$V \rightarrow w$

where *V* is a single nonterminal symbol, and *w* is a string of terminals and/or nonterminals (*w* can be empty).

Context-free grammars play a central role in the description and design of programming languages and compilers. They are also used for analyzing the syntax of natural languages. Noam Chomsky has posited that all human languages are based on context free grammars at their core, with additional processes that can manipulate the output of the context free component.

Formal Definition:

A context-free grammar *G* is defined by 4-tuple: $G = (V, \Sigma, R, S)$ where

- V is a finite set of *non-terminal* characters or variables. They represent different types of phrase or clause in the sentence. They are sometimes called syntactic categories. Each variable represents a language.

- Σ is a finite set of *terminal*s, disjoint from V, which make up the actual content of the sentence. The set of terminals is the alphabet of the language defined by the grammar.

- R is a relation from V to $(V \cup \Sigma)^*$ such that $\exists w \in (V \cup \Sigma)^* : (S, w) \in R$
These relations called productions or rewrite rules.

- S is the start variable (or start symbol), used to represent the whole sentence (or program). It must be an element of V.

The language of a grammar is $G = (V, \Sigma, R, S)$ the set

$$L(G) = \{w \in \Sigma^* : S \overset{*}{\Rightarrow} w\}$$

A language L is said to be a context-free language (CFL) if there exists a CFG, G such that $L = L(G)$.

Example: The CFG for balanced parentheses can be defined by following production rules:

S → SS
S → (S)
S → ()

There are two terminal symbols (and) and one nonterminal symbol S. The first rule allows S's to multiply, the second rule allows S's to become enclosed by matching parentheses, and the third rule terminates the recursion.

Starting with S, applying the rules, one can construct:

S → SS → SSS → (S)SS → ((S))SS → ((SS))S(S)
→ (((()S))S(S) → (((())))S(S) → (((()))))()(S)
→ (((()))))()(())

Regular Grammar (Type 1):

In regular grammars, the left hand side is only a single nonterminal symbol, the right side may be the empty string, or a single terminal symbol, or a single terminal symbol followed by a nonterminal symbol, but nothing else. (Sometimes a broader definition is used: one can allow longer strings of terminals or single nonterminals without anything else, making languages easier to denote while still defining the same class of languages.)

A regular grammar can be either right regular or left regular. A **right regular grammar** (also called right linear grammar) is a formal grammar (*N*, Σ, *P*, *S*) such that all the production rules in *P* are of one of the following forms:

1. *B* → *a* - where *B* is a non-terminal in *N* and *a* is a terminal in Σ
2. *B* → *aC* - where *B* and *C* are in *N* and *a* is in Σ
3. *B* → ε - where *B* is in *N* and ε denotes the empty string, i.e. the string of length 0.

In a **left regular grammar** (also called left linear grammar), all rules obey the forms

1. $A \rightarrow a$ - where A is a non-terminal in N and a is a terminal in Σ
2. $A \rightarrow Ba$ - where A and B are in N and a is in Σ
3. $A \rightarrow \varepsilon$ - where A is in N and ε is the empty string.

Example:

An example of a right regular grammar G with $N = \{S, A\}$, $\Sigma = \{a, b, c\}$, P consists of the following rules

$S \rightarrow aS$
$S \rightarrow bA$
$A \rightarrow \varepsilon$
$A \rightarrow cA$

and S is the start symbol. This grammar describes the same language as the regular expression a*bc*.

Physical Symbol System (PSS):

The **physical symbol system hypothesis** (PSSH), first formulated by **Newell and Simon**, states that **"a physical symbol system [such as a digital computer, for example] has the necessary and sufficient means for intelligent action." The hypothesis implies that computers, when we provide them with the appropriate symbol-processing programs, will be capable of intelligent action.** It also implies, as Newell and Simon wrote, that "the symbolic behavior of man arises because he has the characteristics of a physical symbol system."

This claim implies both that human thinking is a kind of symbol manipulation (because a symbol system is necessary for intelligence) and that machines can be intelligent (because a symbol system is sufficient for intelligence).

A PSS consists of

- Symbols - set of entities that are physical patterns
- Symbol Structures - number of instances/tokens related in some physical way
- Processes – operate on these expressions to produce other expressions

A natural question to ask about symbols and representation is *what is a symbol?* **Allen Newell** considered this question in *Unified Theories of Cognition*. He differentiated between *symbols* (the phenomena in the abstract) and *tokens* (their physical instantiations). Tokens "stood for" some larger concept. They could be manipulated locally until the information in the larger concept was needed, when local processing would have to stop and access the distal site where the information was stored. The distal information may itself be symbolically encoded, potentially leading to a graph of distal accesses for information.

Newell defined *symbol systems* according to their characteristics. Firstly, they may form a universal computational system. They have

- *memory* to contain the distal symbol information,
- *symbols* to provide a pattern to match or index distal information,
- *operations* to manipulate symbols,
- *interpretation* to allow symbols to specify operations, and,
- *capacities* for:
    1. sufficient memory,
    2. composability (that the operators may make any symbol structure),
    3. interpretability (that symbol structures be able to encode any meaningful arrangement of operations).

Finally, Newell defined symbolic architectures as *the fixed structure that realizes a symbol system*. The fixity implies that the behavior of structures on top of it (i.e. "programs") mainly depend upon the details of the symbols, operations and interpretations at the symbol system level, not upon how the symbol system (and its components) are implemented. How well this ideal hold is a measure of the *strength* of that level.

Here is Simon's own description:

A physical symbol system holds a set of entities, called symbols. These are physical patterns (e.g., chalk marks on a blackboard) that can occur as components of symbol structures. In the case of computers, a symbol system also possesses a number of simple processes that operate upon symbol structures - processes that create, modify, copy and destroy symbols. A physical symbol system is a machine that, as it moves through time, produces an evolving collection of symbol structures. Symbol structures can, and commonly do, sever as internal representations (e.g., "mental images") of the environment to which the symbol system is seeking to adapt. They allow it to model that environment with greater or less veridicality and in greater or less detail, and consequently to reason about it. Of course, for this capability to be of any use to the symbol system, it must have windows on the world and hands, too. It must have means for acquiring information from the external environment that can be encoded into internal symbols, as well as means for producing symbols that initiate action upon the environment. Thus it must use symbols to designate objects and relations and actions in the world external to the system,

Symbols may also designate processes that the symbol system can interpret and execute. Hence the program that governs the behavior of a symbol system can be stored, along with other symbol structures, in the system's own memory, and executed when activated.

Examples of Symbol Systems

System	**Symbol**	**Expressions**	**Processes**
Logic	And, Or, Not,	Propositions (T/F)	Rules of Inference
Algebra	+, -, *, /, y, z, 1, 2, 3….	Equations (2+3=5)	Rules of Algebra
Digital Computer	0, 1	00000111000…..	Program
Chess	Chess Pieces	Position of pieces on the board	Legal Chess Moves
Brain	Encoded in brain	Thoughts	Mental operations like thinking

The advantages of *symbolic* architectures are:

1. much of human knowledge is symbolic, so encoding it in a computer is more straight-forward;
2. how the architecture reasons may be analogous to how humans do, making it easier for humans to understand;
3. they may be made computationally complete (e.g. Turing Machines).

These advantages have been considered as one of the fundamental tenets of artificial intelligence known as the **physical symbol system hypothesis**. The hypothesis proposes that a physical symbol system has the necessary and sufficient means for general intelligence.

[Unit -4: Approaches]

Introduction to Cognitive Science (CSC-255)

Central Department of Computer Science & Information Technology
Tribhuvan University

Unit 4: Approaches

Connectionism:

Connectionism is a set of approaches in the fields of artificial intelligence, cognitive psychology, cognitive science, neuroscience and philosophy of mind, that models mental or behavioral phenomena as the emergent processes of *interconnected networks of simple units*. There are many forms of connectionism, but the most common forms use neural network models. **Connectionism** is the name for the computer modeling approach to information processing based on the design or architecture of the brain. **Neural networks** are by far the most commonly used connectionist model today

In most connectionist models, networks change over time. A closely related and very common aspect of connectionist models is *activation*. At any time, a unit in the network has an activation, which is a numerical value intended to represent some aspect of the unit. For example, if the units in the model are neurons, the activation could represent the probability that the neuron would generate an action potential spike. If the model is a spreading activation model, then over time a unit's activation spreads to all the other units connected to it. Spreading activation is always a feature of neural network models, and it is very common in connectionist models used by cognitive psychologists.

Connectionist Models: Refer unit 3 Neural Network.

Learning in Neural Networks:

Learning: One of the powerful features of neural networks is learning. **Learning in neural networks is carried out by adjusting the connection weights among neurons**. It is similar to a biological nervous system in which learning is carried out by changing synapses connection strengths, among cells.

The operation of a neural network is determined by the values of the interconnection weights. There is no algorithm that determines how the weights should be assigned in order to solve specific problems. Hence, the weights are determined by a learning process

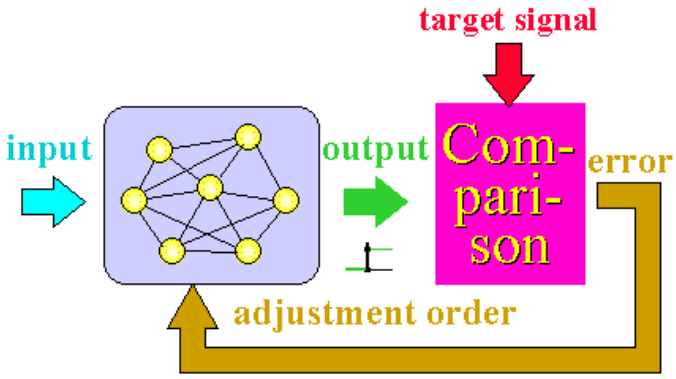
Learning may be classified into two categories:

(1) Supervised Learning
(2) Unsupervised Learning

Supervised Learning:

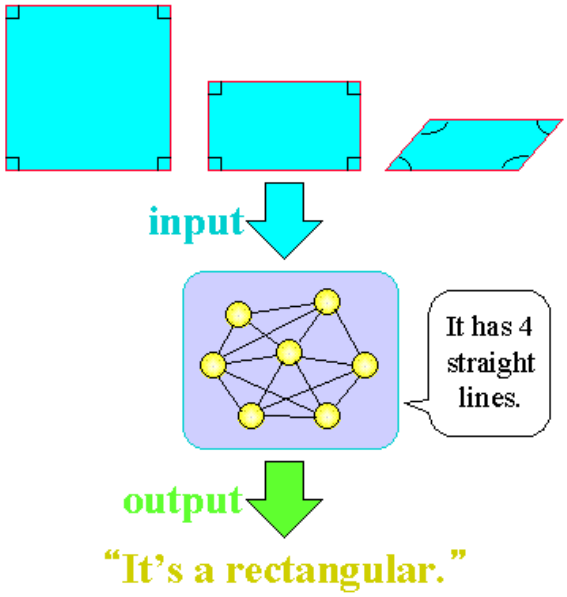
In supervised learning, the network is presented with inputs together with the target (teacher signal) outputs. Then, the neural network tries to produce an output as close as possible to the target signal by adjusting the values of internal weights. The most common supervised learning method is the “error correction method”.

Error correction method is used for networks which their neurons have discrete output functions. Neural networks are trained with this method in order to reduce the error (difference between the network's output and the desired output) to zero.



Unsupervised Learning:

In unsupervised learning, there is no teacher (target signal) from outside and the network adjusts its weights in response to only the input patterns. A typical example of unsupervised learning is **Hebbian learning.**



Consider a machine (or living organism) which receives some sequence of inputs x1, x2, x3, . . ., where xt is the sensory input at time t. In supervised learning the machine is given a sequence of input & a sequence of desired outputs y1, y2, . . . , and the goal of the machine is to learn to produce the correct output given a new input. While, in unsupervised learning the machine simply receives inputs x1, x2, . . ., but obtains neither supervised target outputs, nor rewards from its environment. It may seem somewhat mysterious to imagine what the machine could possibly learn given that it doesn't get any feedback from its environment. However, it is possible to develop of formal framework for unsupervised learning based on

the notion that the machine's goal is to build representations of the input that can be used for decision making, predicting future inputs, efficiently communicating the inputs to another machine, etc. In a sense, unsupervised learning can be thought of as finding patterns in the data above and beyond what would be considered pure unstructured noise.

Hebbian Learning:

The oldest and most famous of all learning rules is Hebb's postulate of learning:

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes take place in one or both cells such that A's efficiency as one of the cells firing B is increased"

From the point of view of artificial neurons and artificial neural networks, Hebb's principle can be described as a method of determining how to alter the weights between model neurons. **The weight between two neurons increases if the two neurons activate simultaneously—and reduces if they activate separately.** Nodes that tend to be either both positive or both negative at the same time have strong positive weights, while those that tend to be opposite have strong negative weights.

Hebb's Algorithm:

Step 0: initialize all weights to 0

Step 1: Given a training input, s, with its target output, t, set the activations of the input units: $x_i = s_i$

Step 2: Set the activation of the output unit to the target value: $y = t$

Step 3: Adjust the weights: $w_i(new) = w_i(old) + x_i y$

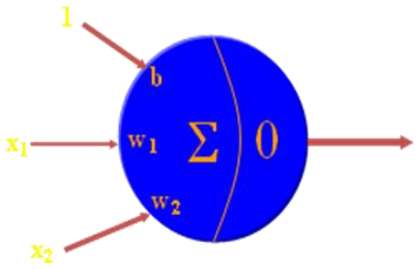
Step 4: Adjust the bias (just like the weights): $b(new) = b(old) + y$

Example:

PROBLEM: Construct a Hebb Net which performs like an AND function, that is, only when both features are "active" will the data be in the target class.

TRAINING SET (with the bias input always at 1):

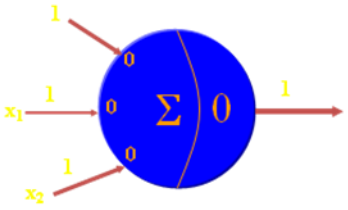
x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



Training-First Input:

- Initialize the weights to 0

Present the first input:
(1 1 1) with a target of 1



Update the weights:

$w_1(new) = w_1(old) + x_1 t$

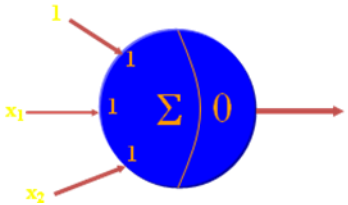
$= 0 + 1 = 1$

$w_2(new) = w_2(old) + x_2 t$

$= 0 + 1 = 1$

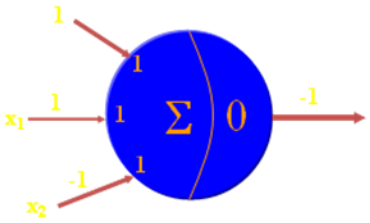
$b(new) = b(old) + t$

$= 0 + 1 = 1$



Training- Second Input:

- **Present the second input:**
  (1 -1 1) with a target of -1



Update the weights:

$w_1(new) = w_1(old) + x_1 t$

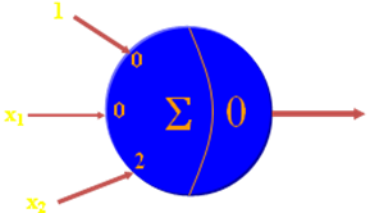
$= 1 + 1(-1) = 0$

$w_2(new) = w_2(old) + x_2 t$

$= 1 + (-1)(-1) = 2$

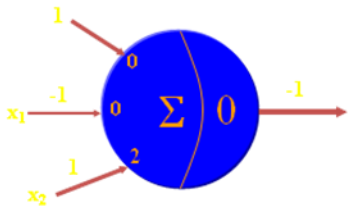
$b(new) = b(old) + t$

$= 1 + (-1) = 0$



Training- Third Input:

- **Present the third input: (-1 1 1) with a target of -1**



Update the weights:

$w_1(new) = w_1(old) + x_1 t$

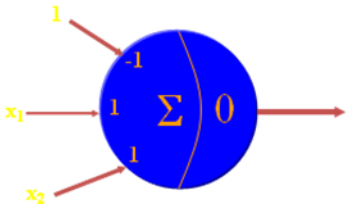
$= 0 + (-1)(-1) = 1$

$w_2(new) = w_2(old) + x_2 t$

$= 2 + 1(-1) = 1$

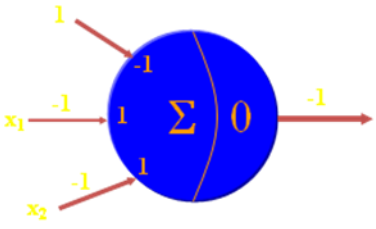
$b(new) = b(old) + t$

$= 0 + (-1) = -1$



Training- Fourth Input:

- **Present the fourth input: (-1 -1 1) with a target of -1**



Update the weights:

$w_1(new) = w_1(old) + x_1 t$

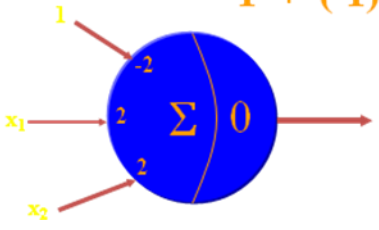
$= 1 + (-1)(-1) = 2$

$w_2(new) = w_2(old) + x_2 t$

$= 1 + (-1)(-1) = 1$

$b(new) = b(old) + t$

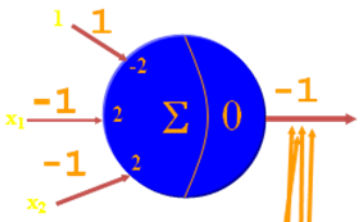
$= -1 + (-1) = -2$



Final Neuron:

- **This neuron works:**

x1	x2	bias	Target
1	1	1	1
1	-1	1	-1
-1	1	1	-1
-1	-1	1	-1



$1*2 + 1*2 + 1*(-2) = 2 > 0$

$(-1)*2 + 1*2 + 1*(-2) = -2 < 0$

$1*2 + (-1)*2 + 1*(-2) = -2 < 0$

$(-1)*2 + (-1)*2 + 1*(-2) = -6 < 0$

Perceptron Learning Theory:

The term "Perceptrons" was coined by Frank RosenBlatt in 1962 and is used to describe the connection of simple neurons into networks. These networks are simplified versions of the real nervous system where some properties are exagerrated and others are ignored. For the moment we will concentrate on Single Layer Perceptrons.

So how can we achieve learning in our model neuron? We need to train them so they can do things that are useful. To do this we must allow the neuron to learn from its mistakes. There is in fact a learning paradigm that achieves this, it is known as supervised learning and works in the following manner.

i. set the weight and thresholds of the neuron to random values.
ii. present an input.
iii. caclulate the output of the neuron.
iv. alter the weights to reinforce correct decisions and discourage wrong decisions, hence reducing the error. So for the network to learn we shall increase the weights on the active inputs when we want the output to be active, and to decrease them when we want the output to be inactive.
v. Now present the next input and repeat steps iii. - v.

Perceptron Learning Algorithm:

The algorithm for Perceptron Learning is based on the supervised learning procedure discussed previously.

Algorithm:

i. Initialize weights and threshold.

Set $w_i(t)$, $(0 <= i <= n)$, to be the weight i at time t, and $ø$ to be the threshold value in the output node. Set w_0 to be $-ø$, the bias, and x_0 to be always 1.

Set $w_i(0)$ to small random values, thus initializing the weights and threshold.

ii. Present input and desired output

Present input $x_0, x_1, x_2, ..., x_n$ and desired output $d(t)$

iii. Calculate the actual output

$$y(t) = g\,[w_0(t)x_0(t) + w_1(t)x_1(t) + ... + w_n(t)x_n(t)]$$

iv. Adapts weights

$w_i(t+1) = w_i(t) + \alpha[d(t) - y(t)]x_i(t)$, where $0 <= \alpha <= 1$ (learning rate) is a positive gain function that controls the adaption rate.

Steps iii. and iv. are repeated until the iteration error is less than a user-specified error threshold or a predetermined number of iterations have been completed.

Please note that the weights only change if an error is made and hence this is only when learning shall occur.

Delta Rule:

The **delta rule** is a gradient descent learning rule for updating the weights of the artificial neurons in a single-layer perceptron. It is a special case of the more general backpropagation algorithm. For a neuron j with activation function $g(x)$ the delta rule for j's ith weight w_{ji} is given by

$$\Delta w_{ji} = \alpha(t_j - y_j)g'(h_j)x_i,$$

where α is a small constant called *learning rate*, $g(x)$ is the neuron's activation function, t_j is the target output, h_j is the weighted sum of the neuron's inputs, y_j is the actual output, and x_i is the ith input. It holds $h_j = \sum x_i w_{ji}$ and $y_j = g(h_j)$.

The delta rule is commonly stated in simplified form for a perceptron with a linear activation function as

$$\Delta w_{ji} = \alpha(t_j - y_j)x_i$$

Backpropagation

It is a supervised learning method, and is an implementation of the **Delta rule**. It requires a teacher that knows, or can calculate, the desired output for any given input. It is most useful for feed-forward networks (networks that have no feedback, or simply, that have no connections that loop). The term is an abbreviation for "backwards propagation of errors". Backpropagation requires that the activation function used by the artificial neurons (or "nodes") is differentiable.

As the algorithm's name implies, the errors (and therefore the learning) propagate backwards from the output nodes to the inner nodes. So technically speaking, backpropagation is used to calculate the gradient of the error of the network with respect to the network's modifiable weights. This gradient is almost always then used in a simple *stochastic gradient descent algorithm, is a general optimization algorithm, but is typically used to fit the parameters of a machine learning model, to find weights that minimize the error*. Often the term "backpropagation" is used in a more general sense, to refer to the entire procedure encompassing both the calculation of the gradient and its use in stochastic gradient descent. Backpropagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited.

Backpropagation networks are necessarily multilayer perceptrons (usually with one input, one hidden, and one output layer). In order for the hidden layer to serve any useful function, multilayer networks must have non-linear activation functions for the multiple layers: a multilayer network using only linear activation functions is equivalent to some single layer, linear network.

Summary of the backpropagation technique:

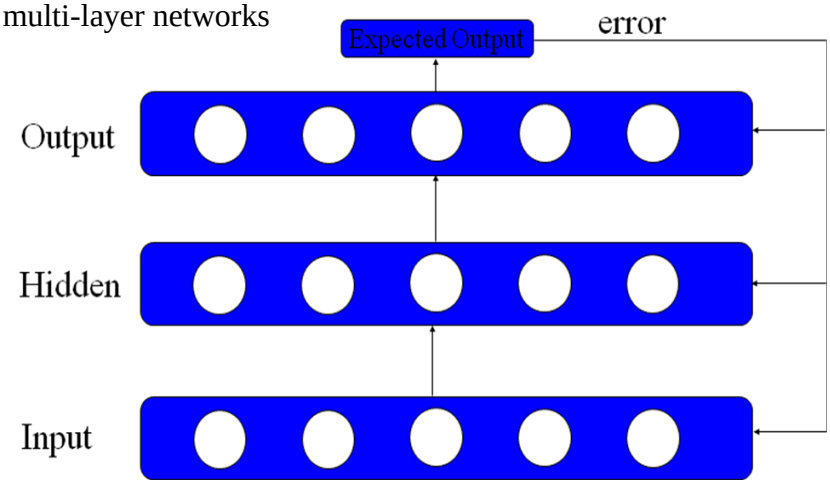
1. Present a training sample to the neural network.
2. Compare the network's output to the desired output from that sample. Calculate the error in each output neuron.
3. For each neuron, calculate what the output should have been, and a *scaling factor*, how much lower or higher the output must be adjusted to match the desired output. This is the local error.
4. Adjust the weights of each neuron to lower the local error.
5. Assign "blame" for the local error to neurons at the previous level, giving greater responsibility to neurons connected by stronger weights.
6. Repeat from step 3 on the neurons at the previous level, using each one's "blame" as its error.

Characteristics:

- A multi-layered perceptron has three distinctive characteristics
  - The network contains one or more layers of hidden neurons
  - The network exhibits a high degree of connectivity
  - Each neuron has a smooth (differentiable everywhere) nonlinear activation function, the most common is the sigmoidal nonlinearity:

$$y_j = \frac{1}{1 + e^{s_j}}$$

- The backpropagation algorithm provides a computational efficient method for training multi-layer networks

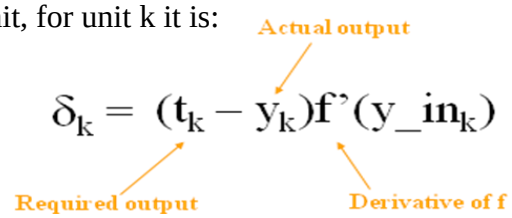


Algorithm:

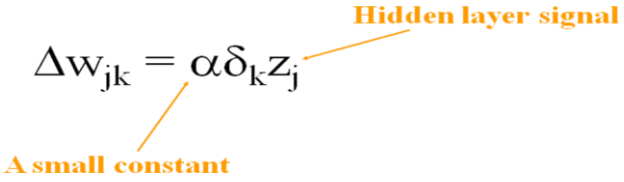
Step 0: Initialize the weights to small random values

Step 1: Feed the training sample through the network and determine the final output

Step 2: Compute the error for each output unit, for unit k it is:



Step 3: Calculate the weight correction term for each output unit, for unit k it is:



Step 4: Propagate the delta terms (errors) back through the weights of the hidden units where the delta input for the j^{th} hidden unit is:

$$\delta\_in_j = \sum_{k=1}^{m} \delta_k w_{jk}$$

The delta term for j^{th} hidden unit is: $\delta_j = \delta\_in_j f'(z\_in_j)$

Step 5: Calculate the weight correction term for the hidden units: $\Delta w_{ij} = \alpha \delta_i x_i$

Step 6: Update the weights: $w_{ik}(new) = w_{ik}(old) + \Delta w_{ik}$

Step 7: Test for stopping (maximum cylces, small changes, etc)

Note: There are a number of options in the design of a backprop system;

- Initial weights – best to set the initial weights (and all other free parameters) to random numbers inside a small range of values (say –0.5 to 0.5)
- Number of cycles – tend to be quite large for backprop systems
- Number of neurons in the hidden layer – as few as possible

Gelernter Response to Descartes:

David Gelernter uses the term "consciousness" to denote the possession of what philosophers call *qualia*. He's not talking about the differences between the brain states of waking and sleeping animals, and he's not talking about self-consciousness -- an animal's ability to recognize itself in a mirror, or to use the states of its own body (including its brain) as subjects for further cognition. Qualia are the *felt character of experience*. To be conscious, in Gelernter's sense, is to have qualia.

Gelernter divides artificial-intelligence theorists into two camps: cognitivists and anticognitivists. Cognitivists believe that, if human beings have qualia, then a robot that behaves exactly like a human being does, too. Gelernter's initial claims:

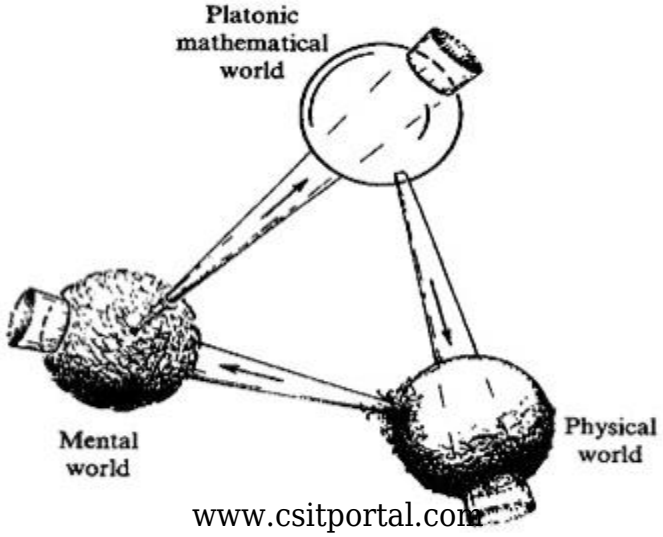
(1) "This subjectivity of mind has an important consequence: *there is no objective way to tell whether some entity is conscious.*"
(2) "we know our fellow humans are conscious."

Human thought, asserts Gelernter, exists along a continuum, spanning from high-focus thinking -- in-depth analytical problem solving -- to low-focus thoughts, consisting of the daydreams and hallucinations that occur when one's mind is wandering. Artificial intelligence research has historically focused on the logical, goal-driven thoughts at the high-end of the spectrum. Gelernter argues that, if the goal truly is to create a computer that can solve problems in a way similar to that of the human mind, then study of unfocused, emotional, low-end thinking must be incorporated into artificial intelligence research, for it is as central to human cognition as logic.

Penrose Response to Descartes:

Roger Penrose has a picture of mind and matter that is not just a relation between logical and physical, but involves three worlds: Platonic, mathematical and physical.

In his more recent *The Road to Reality*, Roger Penrose leaves aside the question of computability in physics, and gets down to the core of physics itself. Once you are past the first thousand pages you can read about his own vision of what it should be like: twistor theory.



Additionally, **Roger Penrose has proposed the idea that the human mind does not use a knowably sound calculation procedure to understand and discover mathematical intricacies.** This would mean that a normal Turing complete computer would not be able to ascertain certain mathematical truths that human minds can.

Penrose presents the argument that human consciousness is non-algorithmic, and thus is not capable of being modeled by a conventional Turing machine-type of digital computer. Penrose hypothesizes that quantum mechanics plays an essential role in the understanding of human consciousness. The collapse of the quantum wavefunction is seen as playing an important role in brain function.

Pinker Response to Descartes:

The mind, for **Steven Pinker** as for almost all other cognitive scientists, is computational. This does *not* mean they think it works just like the computer you're reading this on, but that has representations, which it transforms in a rule-governed, algorithmic way. Moreover, the mind is not a single, general-purpose computer, but a collection of them, of "mental modules" or "mental organs," specialized as to subject matter, each with its own particular learning mechanism ("an instinct to acquire an art," in a phrase Pinker lifts from Darwin). This modularity is evident in studying how children learn, and also from tracing the effects of brain lesions which, if sufficiently localized, impair specific abilities depending on where the brain is hurt, and leave others intact. Just as, baring developmental defects, wounds, or the ravages of disease, all human beings have the same physical organs, we all have the same mental organs, whose general structure is, again, the same from person to person.

By insisting on the complexity of the mind, Pinker claims that;

- thinking is a kind of computation used to work with configurations of symbols,
- the mind is organized into specialized modules or mental organs,
- the basic logic of the modules is contained in our genetic program,
- that natural selection shaped these operations to facilitate replication of genes into the next generation

Pinker thus shows that the computational model of mind is highly significant because it has solved not only philosophical problems, but also started the computer revolution, posed important neuroscience questions, and provided psychology with a very valuable research agenda

Searle Response to Descartes:

Consciousness is a biological phenomenon. We should think of consciousness as part of our ordinary biological history, along with digestion, growth, mitosis and meiosis. However, though consciousness is a biological phenomenon, it has some important features that other biological phenomena do not have. The most important of these is what I (John Searle) have called its `subjectivity'. There is a sense in which each person's consciousness is private to

that person, a sense in which he is related to his pains, tickles, itches, thoughts and feelings in a way that is quite unlike the way that others are related to those pains, tickles, itches, thoughts and feelings. This phenomenon can be described in various ways. It is sometimes described as that feature of consciousness by way of which there is something that it's like or something that it feels like to be in a certain conscious state. If somebody asks me what it feels like to give a lecture in front of a large audience I (Searle) can answer that question. But if somebody asks what it feels like to be a shingle or a stone, there is no answer to that question because shingles and stones are not conscious. The point is also put by saying that conscious states have a certain qualitative character; the states in question are sometimes described as **'qualia'**.

In spite of its etymology, consciousness should not be confused with knowledge, it should not be confused with attention, and it should not be confused with self-consciousness. I (Searle) will consider each of these confusions in turn.

Conscious states are caused by lower level neurobiological processes in the brain and are themselves higher level features of the brain. The key notions here are those of *cause* and *feature*. As far as we know anything about how the world works, variable rates of neuron firings in different neuronal architectures cause all the enormous variety of our conscious life. All the stimuli we receive from the external world are converted by the nervous system into one medium, namely, variable rates of neuron firings at synapses. And equally remarkably, these variable rates of neuron firings cause all of the colour and variety of our conscious life. The smell of the flower, the sound of the symphony, the thoughts of theorems in Euclidian geometry -- all are caused by lower level biological processes in the brain; and as far as we know, the crucial functional elements are neurons and synapses.

The first step in the solution of the mind-body problem is: brain processes *cause* conscious processes. This leaves us with the question, what is the ontology, what is the form of existence, of these conscious processes? More pointedly, does the claim that there is a causal relation between brain and consciousness commit us to a dualism of `physical' things and `mental' things? The answer is a definite no. Brain processes cause consciousness but the consciousness they cause is not some extra substance or entity. It is just a higher level feature of the whole system. The two crucial relationships between consciousness and the brain, then, can be summarized as follows: lower level neuronal processes in the brain cause consciousness and consciousness is simply a higher level feature of the system that is made up of the lower level neuronal elements.

John Searle has offered a thought experiment known as the **Chinese Room** that demonstrates this problem. Imagine that there is a man in a room with no way of communicating to anyone or anything outside of the room except for a piece of paper that is passed under the door. With the paper, he is to use a series of books provided to decode and "answer" what is on the paper. The symbols are all in Chinese, and all the man knows is where to look in the books, which then tell him what to write in response. It just so happens that this generates a conversation that the Chinese man outside of the room can actually understand, but can our man in the room really be said to understand it? This is essentially what the computational theory of mind presents us with; a model in which the mind simply

decodes symbols and outputs more symbols. It is argued that perhaps this is not real learning or thinking at all. However, it can be argued in response to this that it is the man and the paper together that understand Chinese, albeit in a rudimentary way due to the rudimentary nature of the system; as opposed to if the man learned Chinese, which would create a sophisticated system of communicating Chinese.

Natural Language Processing:

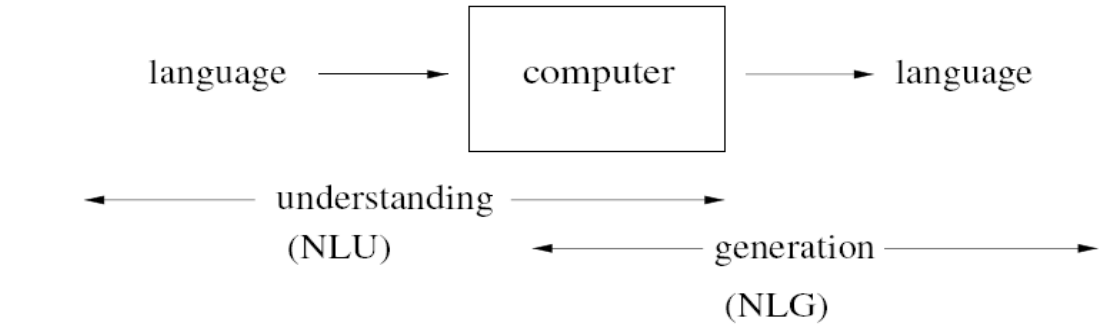
Perception and communication are essential components of intelligent behavior. They provide the ability to effectively interact with our environment. Humans perceive and communicate through their five basic senses of sight, hearing, touch, smell and taste, and their ability to generate meaningful utterances. Developing programs that understand a natural language is a difficult problem. Natural languages are large. They contain infinity of different sentences. No matter how many sentences a person has heard or seen, new ones can always be produced. Also, there is much ambiguity in a natural language. Many words have several meanings and sentences can have different meanings in different contexts. This makes the creation of programs that understand a natural language, one of the most challenging tasks in AI.

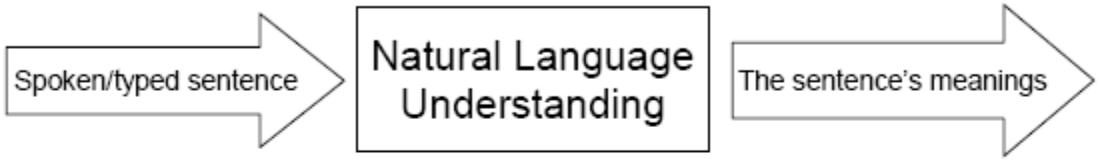
Developing programs to understand natural language is important in AI because a natural form of communication with systems is essential for user acceptance. AI programs must be able to communicate with their human counterparts in a natural way, and natural language is one of the most important mediums for that purpose. So, Natural Language Processing (NLP) is the field that deals with the computer processing of natural languages, mainly evolved by people working in the field of Artificial Intelligence.

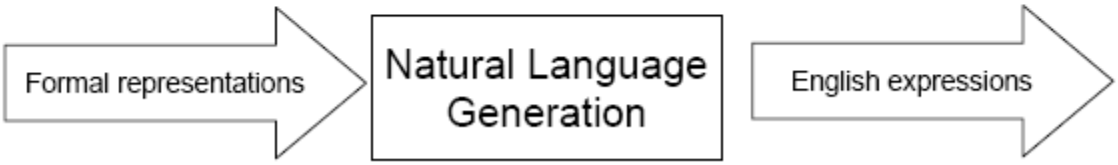
Natural Language Processing (NLP), is the attempt to extract the fuller meaning representation from the free text. Natural language processing is a technology which involves converting spoken or written human language into a form which can be processed by computers, and vice versa. Some of the better-known applications of NLP include:

- **Voice recognition software** which translates speech into input for word processors or other applications;
- **Text-to-speech synthesizers** which read text aloud for users such as the hearing-impaired;
- **Grammar and style checkers** which analyze text in an attempt to highlight errors of grammar or usage;
- **Machine translation systems** which automatically render a document such as a web page in another language.

computers using natural language as input and/or output







Natural Language Generation:

"Natural Language Generation (NLG), also referred to as text generation, is a subfield of natural language processing (NLP; which includes computational linguistics)

Natural Language Generation (NLG) is the natural language processing task of generating natural language from a machine representation system such as a knowledge base or a logical form.

In a sense, one can say that an NLG system is like a translator that converts a computer based representation into a natural language representation. However, the methods to produce the final language are very different from those of a compiler due to the inherent expressivity of natural languages.

NLG may be viewed as the opposite of natural language understanding. The difference can be put this way: whereas in natural language understanding the system needs to disambiguate the input sentence to produce the machine representation language, in NLG the system needs to make decisions about how to put a concept into words.

The different types of generation techniques can be classified into four main categories:

- Canned text systems constitute the simplest approach for single-sentence and multi-sentence text generation. They are trivial to create, but very inflexible.
- Template systems, the next level of sophistication, rely on the application of pre-defined templates or schemas and are able to support flexible alterations. The template approach is used mainly for multi-sentence generation, particularly in applications whose texts are fairly regular in structure.
- Phrase-based systems employ what can be seen as generalized templates. In such systems, a phrasal pattern is first selected to match the top level of the input, and then each part of the pattern is recursively expanded into a more specific phrasal pattern that matches some subportion of the input. At the sentence level, the phrases resemble phrase structure grammar rules and at the discourse level they play the role of text plans.
- Feature-based systems, which are as yet restricted to single-sentence generation, represent each possible minimal alternative of expression by a single feature. Accordingly, each sentence is specified by a unique set of features. In this framework, generation consists in the incremental collection of features appropriate for each portion of the input. Feature collection itself can either be based on unification or on the traversal of a feature selection network. The expressive power of the approach is very high since any distinction in language can be added to the system as a feature. Sophisticated feature-based generators, however, require very complex input and make it difficult to maintain feature interrelationships and control feature selection.

Many natural language generation systems follow a hybrid approach by combining components that utilize different techniques.

Natural Language Understanding:

Developing programs that understand a natural language is a difficult problem. Natural languages are large. They contain infinity of different sentences. No matter how many sentences a person has heard or seen, new ones can always be produced. Also, there is much ambiguity in a natural language. Many words have several meaning such as can, bear, fly, bank etc, and sentences have different meanings in different contexts.

Example :- a *can* of juice. I *can* do it.

This makes the creation of programs that understand a natural language, one of the most challenging tasks in AI. Understanding the language is not only the transmission of words. It also requires inference about the speakers' goal, knowledge as well as the context of the interaction. We say a program understand natural language if it behaves by taking the correct

or acceptable action in response to the input. A word functions in a sentence as a part of speech. Parts of the speech for the English language are nouns, pronouns, verbs, adjectives, adverbs, prepositions, conjunctions and interjections. Three major issues involved in understanding language.

- A large amount of human knowledge is assumed.
- Language is pattern based, phonemes are components of the words and words make phrases and sentences. Phonemes, words and sentences order are not random.
- Language acts are the product of agents (human or machine).

Levels of knowledge used in Language Understanding

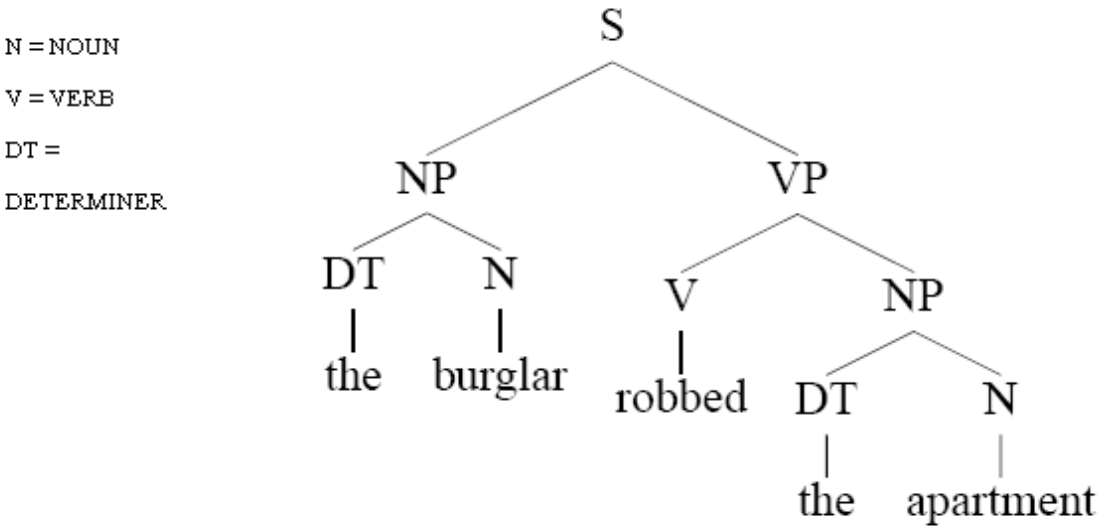
A language understanding knowledge must have considerable knowledge about the structures of the language including what the words are and how they combine into phrases and sentences. It must also know the meanings of the words and how they contribute to the meanings of the sentence and to the context within which they are being used. The component forms of knowledge needed for an understanding of natural languages are sometimes classified according to the following levels.

- **Phonological**
    - Relates sound to the words we recognize. A phoneme is the smallest unit of the sound. Phones are aggregated to the words.
- **Morphological**
    - This is lexical knowledge which relates to the word construction from basic units called morphemes. A morpheme is the smallest unit of meaning. Eg:- *friend* + ly = friendly
- **Syntactic**
    - This knowledge relates to how words are put together or structure red together to form grammatically correct sentences in the language.
- **Semantic**
    - This knowledge is concerned with the meanings of words and phrases and how they combine to form sentence meaning.
- **Pragmatic**
    - This is high – level knowledge which relates to the use of sentences in different contexts and how the context affects the meaning of the sentence.

- **World**
  - Includes the knowledge of the physical world, the world of human social interaction, and the roles of goals and intentions in communication.

Basic Parsing Techniques

Before the meaning of a sentence can be determined, the meanings of its constituent parts must be established. This requires knowledge of the structure of the sentence, the meaning of the individual words and how the words modify each other. The process of determining the syntactical structure of a sentence is known as parsing. Parsing is the process of analyzing a sentence by taking it apart word – by – word and determining its structure from its constituent parts and sub parts. The structure of a sentence can be represented with a syntactic tree. When given an input string, the lexical parts or terms (root words), must first be identified by type and then the role they play in a sentence must be determined. These parts can be combined successively into larger units until a complete tree has been computed.



Noun Phrases (NP): "the burglar", "the apartment"

Verb Phrases (VP): "robbed the apartment"

Sentences (S): "the burglar robbed the apartment"

To determine the meaning of a word, a parser must have access to a lexicon. When the parser selects the word from the input stream, it locates the world in the lexicon and obtains the word's possible functions and features, including the semantic information.

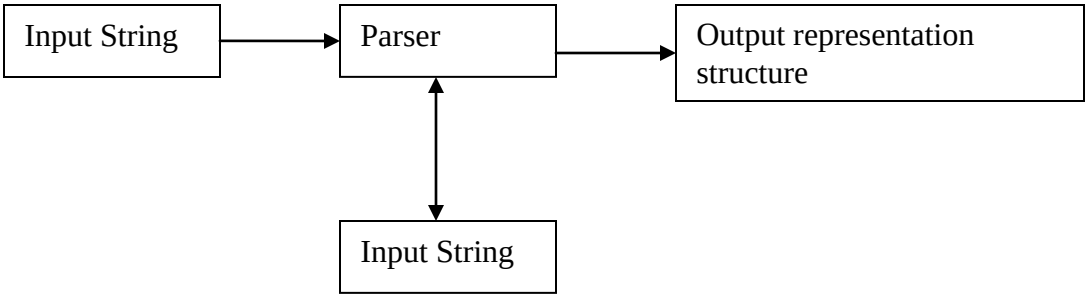


Figure :- Parsing an input to create an output structure

Lexeme (Lexicon) & word forms:

The distinction between these two senses of "word" is arguably the most important one in morphology. The first sense of "word", the one in which *dog* and *dogs* are "the same word", is called a lexeme. The second sense is called *word form*. We thus say that *dog* and *dogs* are different forms of the same lexeme. *Dog* and *dog catcher*, on the other hand, are different lexemes, as they refer to two different kinds of entities. The form of a word that is chosen conventionally to represent the canonical form of a word is called a lemma, or citation form.

A lexicon defines the words of a language that a system knows about. This is includes common words and words that are specific to the domain of the application. Entries include meanings for each word and its syntactic and morphological behavior.

Morphology:

Morphology is the identification, analysis and description of the structure of words (words as units in the lexicon are the subject matter of lexicology). While words are generally accepted as being (with clitics) the smallest units of syntax, it is clear that in most (if not all) languages, words can be related to other words by rules. For example, English speakers recognize that the words *dog*, *dogs*, and *dog catcher* are closely related. English speakers recognize these relations from their tacit knowledge of the rules of word formation in English. They infer intuitively that *dog* is to *dogs* as *cat* is to *cats*; similarly, *dog* is to *dog catcher* as *dish* is to *dishwasher* (in one sense). The rules understood by the speaker reflect specific patterns (or regularities) in the way words are formed from smaller units and how those smaller units interact in speech. In this way, morphology is the branch of linguistics that studies patterns of word formation within and across languages, and attempts to formulate rules that model the knowledge of the speakers of those languages.

Morphological analysis is the process of recognizing the suffixes and prefixes that have been attached to a word.

We do this by having a table of affixes and trying to match the input as:
prefixes +root + suffixes.

- For example: adjective + ly -> adverb. E.g.: [Friend + ly]=friendly
- We may not get a unique result.
- "-s, -es" can be either a plural noun or a 3ps verb
- "-d, -ed" can be either a past tense or a perfect participle

Morphological Information:

- Transform part of speech
  - *green, greenness (adjective, noun)*
  - *walk, walker (verb, noun)*

- Change features of nouns
  - *boat, boats (singular, plural)*

- Bill slept , Bill's bed
  - (subjective case, possessive case)

- Change features of verbs
  - Aspect
    - *I walk. I am walking.* (present, progressive)
  - Tense
    - *I walked. I will walk. I had been walking. (past, future, past progressive)*
  - Number and person
    - *I walk. They walk. (first person singular, third person plural)*

Syntactic Analysis:

Syntactic analysis takes an input sentence and produces a representation of its grammatical structure. A grammar describes the valid parts of speech of a language and how to combine them into phrases. The grammar of English is nearly context free.

A computer grammar specifies which sentences are in a language and their parse trees. A parse tree is a hierarchical structure that shows how the grammar applies to the input. Each level of the tree corresponds to the application of one grammar rule.

It is the starting point for working out the meaning of the whole sentence. Consider the following two sentences.

1. "The dog ate the bone."
2. "The bone was eaten by the dog."

Understanding the structure (via the syntax rules) of the sentences help us work out that it's the bone that gets eaten and not the dog. Syntactic analysis determines possible grouping of words in a sentence. In other cases there may be many possible groupings of words. Consider the sentence "John saw Mary with a telescope". Two different readings based on the groupings.

1. John saw (Mary with a telescope).
2. John (saw Mary with a telescope).

A sentence is syntactically ambiguous if there are two or more possible groupings. Syntactic analysis helps determining the meaning of a sentence by working out possible word structure. Rules of syntax are specified by writing a *grammar* for the language. A parser will check if a sentence is correct according to the grammar. It returns a representation (parse tree) of the sentence's structure. A grammar specifies allowable sentence structures in terms of basic categories such as noun and verbs. A given grammar, however, is unlikely to cover all possible grammatical sentences. Parsing sentences is to help determining their meanings, not just to check that they are correct. Suppose we want a grammar that recognizes sentences like the following.

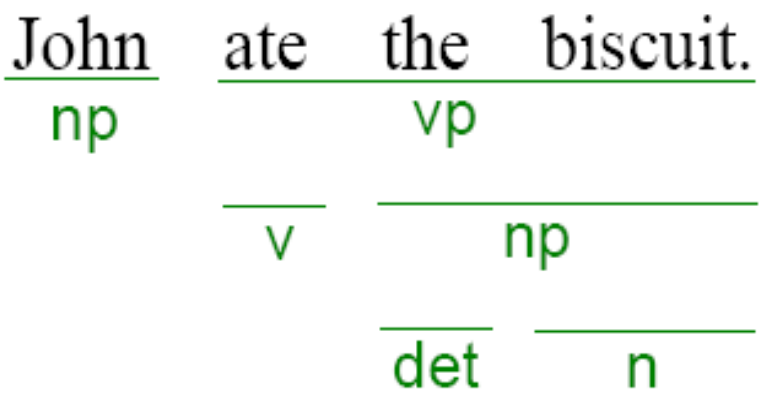
John ate the biscuit.
The lion ate the zebra.
The lion kissed John

But reject incorrect sentences such as

Ate John biscuit the.
Zebra the lion the ate.
Biscuit lion kissed.

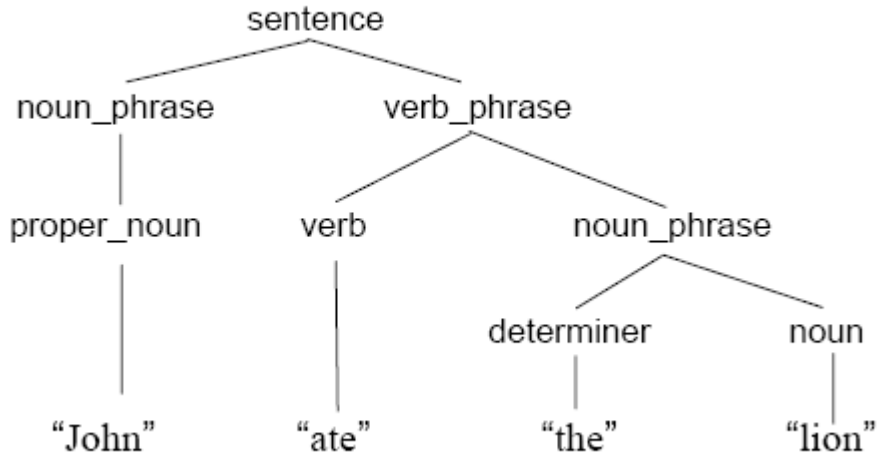
A simple grammar that deals with this is given below

```
sentence --> noun_phase, verb phrase.
noun_phrase --> proper_noun.
noun_phrase --> determiner, noun.
verb_phrase --> verb, noun_phrase.
proper_noun --> [mary].
proper_noun --> [john].
noun --> [zebra].
noun --> [biscuit].
verb --> [ate].
verb --> [kissed].
determiner --> [the].
```



Incorrect sentences like "biscuit lion kissed" will be excluded by the grammar.

- A ***parse trees*** illustrates the syntactic structure of the sentence.



Semantic Analysis:

Semantic analysis is a process of converting the syntactic representations into a meaning representation.

This involves the following tasks:

- Word sense determination
- Sentence level analysis
- Knowledge representation

- Word sense

Words have different meanings in different contexts.

Mary had a bat in her office.

- bat = `a baseball thing'

- bat = `a flying mammal'

- Sentence level analysis

Once the words are understood, the sentence must be assigned some meaning

I saw an astronomer with a telescope.

- Knowledge Representation

Understanding language requires lots of knowledge**.**

- Using predicate logic, for example, one can represent sentences like

"John likes Mary" `likes(john,mary)`

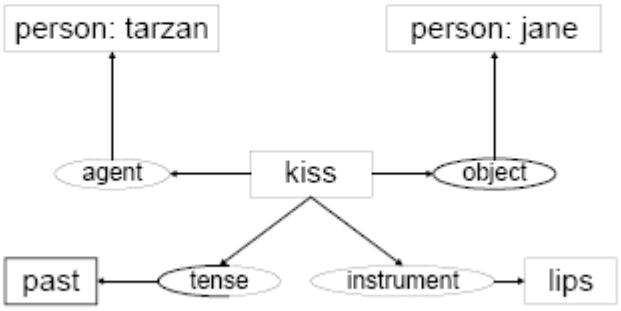
"The man likes Mary" `man(m1)∧likes(m1,mary)`

"A man likes Mary" `∃X(man(X)∧likes(X,mary)`

"A tall bearded man likes Mary"

`∃X(man(X)∧tall(X)∧bearded(X)∧likes(X,mary)`

- Using semantic net, one can represent sentence "Tarzan kissed Jane" as



Parameters in Natural Language Processing:

- Auditory Inputs
- Segmentation
- Syntax Structure
- Semantic Structure
- Pragmatic Analysis

- Auditory Inputs:

Three of our five senses – sight, hearing and touch – are used as *major inputs*. These are usually referred to as the visual, auditory and tactile inputs respectively. They are sometimes called input channels; however, as previously mentioned, the term "channel" is used in various ways, so I will avoid it.

In the fashion of video devices, audio devices are used to either capture or create sound. In some cases, an audio output device can be used as an input device, in order to capture produced sound.

- Microphone
- MIDI keyboard or other digital musical instrument

- Segmentation:

Text segmentation is the process of dividing written text into meaningful units, such as words, sentences, or topics. The term applies both to mental processes used by humans when reading text, and to artificial processes implemented in computers, which are the subject of natural language processing. The problem is non-trivial, because while some written languages have explicit word boundary markers, such as the word spaces of written English and the distinctive initial, medial and final letter shapes of Arabic, such signals are sometimes ambiguous and not present in all written languages.

Word segmentation is the problem of dividing a string of written language into its component words. In English and many other languages using some form of the Latin alphabet, the space is a good approximation of a word delimiter. (Some examples where the space character alone may not be sufficient include contractions like *can't* for *can not*.)

However the equivalent to this character is not found in all written scripts, and without it word segmentation is a difficult problem. Languages which do not have a trivial word segmentation process include Chinese, Japanese, where sentences but not words are delimited, and Thai, where phrases and sentences but not words are delimited.

In some writing systems however, such as the Ge'ez script used for Amharic and Tigrinya among other languages, words are explicitly delimited (at least historically) with a non-whitespace character.

Word splitting is the process of parsing concatenated text (i.e. text that contains no spaces or other word separators) to infer where word breaks exist.

Sentence segmentation is the problem of dividing a string of written language into its component sentences. In English and some other languages, using punctuation, particularly the full stop character is a reasonable approximation. However, even in English this problem is not trivial due to the use of the full stop character for abbreviations, which may or may not also terminate a sentence. For example *Mr.* is not its own sentence in "*Mr. Smith went to the shops in Jones Street."* When processing plain text, tables of abbreviations that contain periods can help prevent incorrect assignment of sentence boundaries. As with word segmentation, not all written languages contain punctuation characters which are useful for approximating sentence boundaries.

Other segmentation problems: Processes may be required to segment text into segments besides words, including morphemes (a task usually called morphological analysis), paragraphs, topics or discourse turns.

A document may contain multiple topics, and the task of computerized text segmentation may be to discover these topics automatically and segment the text accordingly. The topic boundaries may be apparent from section titles and paragraphs. In other cases one needs to use techniques similar to those used in document classification. Many different approaches have been tried.

- Syntax Structure:

Same concept as in the syntactic analysis above

- Semantic Structure:

Same concept as in the semantic analysis above

- Pragmatic Analysis:

This is high level knowledge which relates to the use of sentences in different contexts and how the context affects the meaning of the sentences. It is the study of the ways in which language is used and its effect on the listener. Pragmatic comprises aspects of meaning that depend upon the context or upon facts about real world.

Pragmatics – Handling Pronouns

Handling pronouns such as “he”, “she” and “it” is not always straight forward. Let us see the following paragraph.

“John buys a new telescope. He sees Mary in the distance. He gets out his telescope. He looks at her through it”.

Here, "*her*" refers to *Mary* who was not mentioned at all in the previous sentences. John's telescope was referred to as "*a new telescope*", "*his telescope*" and "*it*".

Let us see one more example

"When is the next flight to Sydney?"
"Does it have any seat left?"

Here, "*it*", refers to a particular flight to Sydney, not Sydney itself.

Pragmatics – Ambiguity in Language

A sentence may have more than one structure such as

"I saw an astronomer with a telescope."

This English sentence has a prepositional phrase "*with a telescope*" which may be attached with either with verb to make phrase "*saw something with telescope*" or to object noun phrase to make phrase "*a astronomer with a telescope*". If we do first, then it can be interpreted as "*I saw an astronomer who is having a telescope*", and if we do second, it can be interpreted as "*Using a telescope I saw an astronomer*".

Now, to remove such ambiguity, one possible idea is that we have to consider the context. If the knowledge base (KB) can prove that whether the telescope is with astronomer or not, then the problem is solved.

Next approach is that; let us consider the real scenario where the human beings communicate. If A says the same sentence "*I saw an astronomer with a telescope*." To B, then in practical, it is more probable that, B (listener) realizes that "*A has seen astronomer who is having a telescope*". It is because, normally, the word "*telescope*" belongs to "*astronomer*", so it is obvious that B realizes so.

If A has says that "*I saw a lady with a telescope*." In this case, B realizes that "*A has seen the lady using a telescope*", because the word "*telescope*" has not any practical relationship with "*lady*" like "*astronomer*".

So, we may be able to remove such ambiguity, by defining a data structure, which can efficiently handle such scenario. This idea may not 100% correct but seemed more probable.

Unit 5
Computer Animation

Introduction

Although we tend to think of **animation** as implying object motions, the term computer animation generally refers to any time sequence of visual changes in a scene. In addition to changing object position with translations or rotations, a computer-generated animation could display time variations in object size, color, transparency, or surface texture.

Some typical applications of computer-generated animation are entertainment (motion pictures and cartoons), advertising, scientific and engineering studies, and training and education. Advertising animations often transition one object shape into another: for example, transforming a can of motor oil into an automobile engine.

Computer animations can also be generated by changing camera parameters, such as position, orientation, and focal length. And we can produce computer animations by charging lighting effects or other parameters and procedures associated with illumination and rendering.

Design of animation sequences

In general, an animation sequence is designed with the following steps:

1. Storyboard layout
2. Object definitions
3. Key-frame specifications
4. Generation of in-between frames

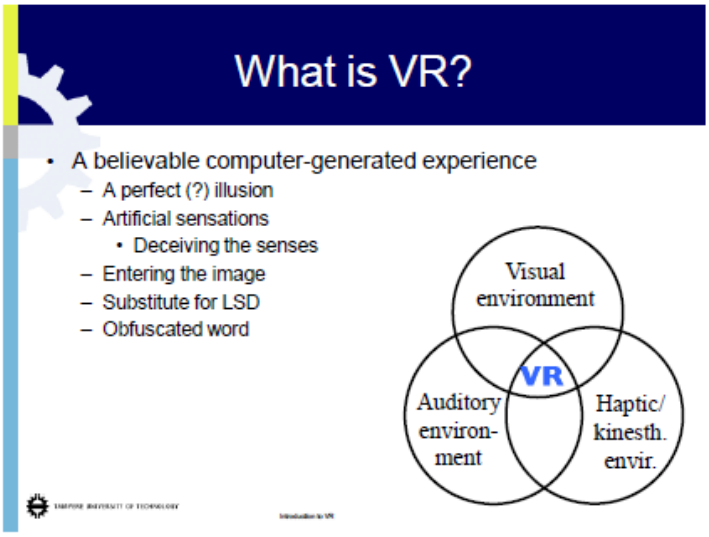
This standard approach for animated cartoons is applied to other animation applications as well, although there are many special applications that do not follow this sequence. Real-time computer animations produced by flight simulators, for instance, display motion sequences in response to settings on the aircraft controls. For frame-by-frame animation, each frame of the scene is separately generated and stored. Later, the frames can be recorded on film or they can be consecutively displayed in "real-time playback" mode.

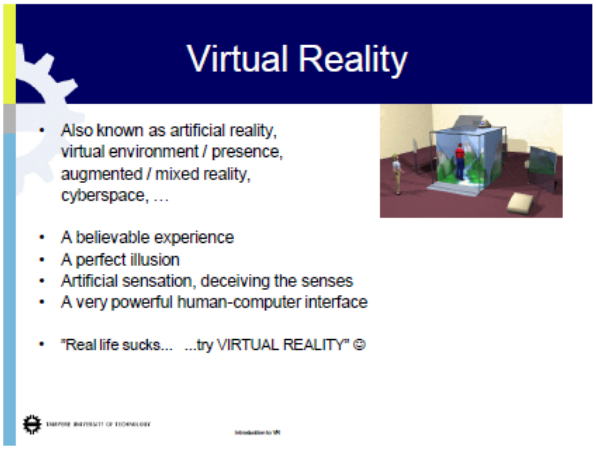
1. **Storyboard** is an outline of the action. It defines the motion sequence as a set of basic events that are to take place. Depending on the type of animation to be produced, the storyboard could consist of a set of rough sketches or it could be a list of the basic ideas for the motion.
2. An **object definition** is given for each participant in the action. Objects can be defined in terms of basic shapes, such as polygons or splines. In addition, the associated movements for each object are specified along with the shape.
3. A **key frame** is a detailed drawing of the scene at a certain time in the animation sequence. Within each key frame, each object is positioned according to the time for that frame. Some key frames are chosen at extreme positions in the action; others are spaced so that the time interval between key frames is not too great. More key frames are specified for intricate motions than for simple, slowly varying motions.
4. **In-betweens** are the intermediate frames between the key frames. The number of in-betweens needed is determined by the media to be used to display the animation. Film requires 24 frames per second, and graphics terminals are refreshed at the rate of 30 to 60 frames per second.

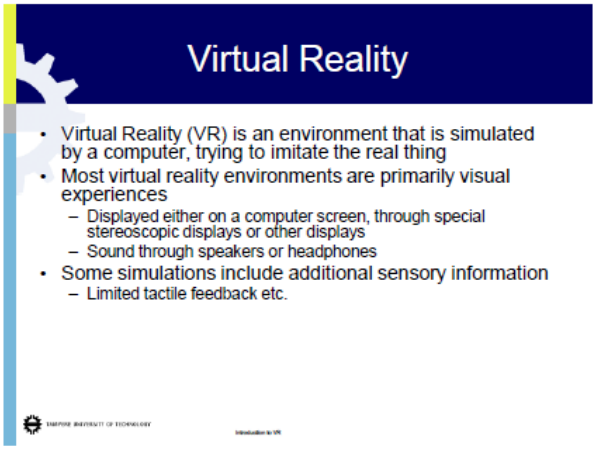
Typically, time intervals for the motion are set up so that there are from three to five in-betweens for each pair of key frames. Depending on the speed specified for the motion, some key frames can be duplicated. For a 1-minute film sequence with no duplication, we would need 1440 frames. With five in-betweens for each pair of key frames, we would need 288 key frames. If the motion is not too complicated, we could space the key frames a little farther apart.

There are several other tasks that may be required, depending on the application. They include motion verification, editing, and production and synchronization of a soundtrack. Many of the functions needed to produce general animations are now computer-generated.

Virtual Reality







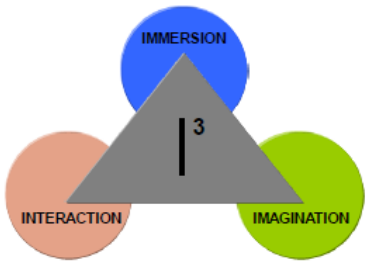
Virtual Reality

- "Virtual reality" originally denoted a fully immersive system
- It has since been used to describe non-orthodox systems lacking wired gloves etc.
- The most immersive experiences I have seen:
  - 3D IMAX (non-VR), Real-D movies (non-VR), CAVE (VR)
  - All of them are very impressive if well done
- In practice, it is very difficult to create a fully convincing virtual reality experience
  - Technical limitations on processing power and image resolution
  - Input/output-devices far from perfect
  - Perfectionism usually not even needed

VR definition

- A simulation in which computer graphics is used to create a realistic-looking world
- Can be a completely synthetic environment without any real counterpart
- Virtual Reality is a high-end user - computer interface that involves real-time simulation and interaction through multiple sensory channels
  - Sensory information may include visual, auditory, haptic, tactile, smell, taste…
  - Visual is dominating

Virtual Reality Triangle



The Three I's of Virtual Reality

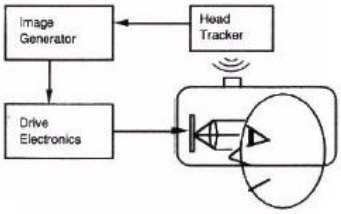
- **Immersion**
  - The feeling of presence, being there
  - The amount and quality of stimuli and sensations
  - Real time: very little latency accepted
    - around 50 ms is a threshold of visual noticability, but varies for all senses
- **Interaction**
  - Not just passive watching
  - Moving in the virtual world
  - Doing all kind of things there
- **Imagination**
  - The applications
  - The ideas
  - The virtual worlds

Properties of VR

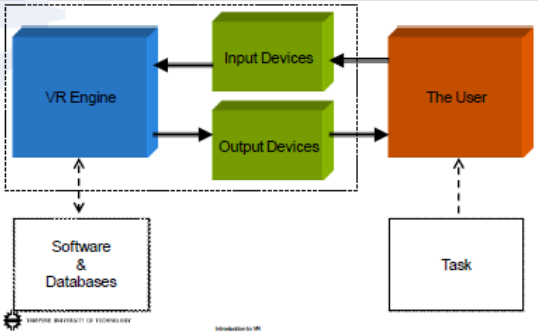
- Synthetically generated environment
  - Computers, 3D, real-time
- Sensory feedback
  - I/O devices
- Interaction, moving
  - In time
  - In space
  - In scale
- Immersion
  - Being there

The Basic Components of VR

- Computing
- Displays (visual, audio, haptics, etc)
- Tracking
- Input



The VR System Architecture



VR Input Devices

- **The ways to transfer information from the user to the computer**
- Mouse, keyboard
- Trackball, joystick, mouse, ...
- Position tracking
- Orientation tracking
- Datagloves
- Exoskeleton (external hardware on hands etc.)
- Data suit
- Motion capture (tracking of body)
- Eye tracking
- Videoanalysis
- Brainwaves (EEG), EMG, EOG
- ...

VR Output Devices

- **Transfering data from the computer to the senses of the user**
- Fooling of ALL(?) senses?
  - Displays
    - Monitors, projectors, HMDs, etc.
  - Localized audio
    - Loudspeakers, headphones
  - Tactile & haptic
    - Force feedback
  - Smell, balance, etc.

Requirements for the System

- Real time, >25 frames/s
  - So the environment is smooth
- Latency < 50 ms
  - Processing power
  - Input devices
- Big data size
  - Enough memory & hard disk space
- 3D speed, as much polygon/s as possible
  - PC: new and powerful graphics cards
  - If not enough power in GPU, fps drops or details drop
- CPU processing power for other calculations & simulations
- Fast networking
- VR may be expensive, or low-cost…