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)

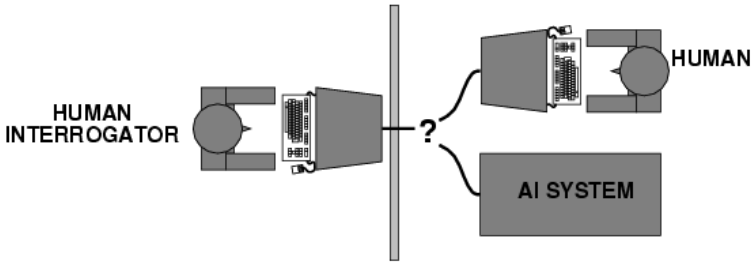
Top dimension is concerned with *thought processes and reasoning*, where as bottom dimension addresses the *behavior*.

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

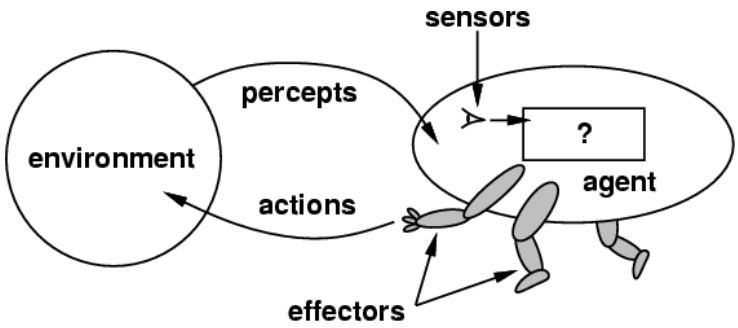
- Autonomous planning and scheduling
- Game playing
- Autonomous Control
- Diagnosis
- Logistics Planning
- Robotics
- Language understanding and problem solving

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.

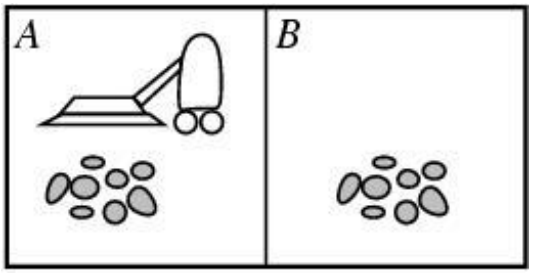
The *agent function* is mathematical concept that maps percept sequence to actions.

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

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?

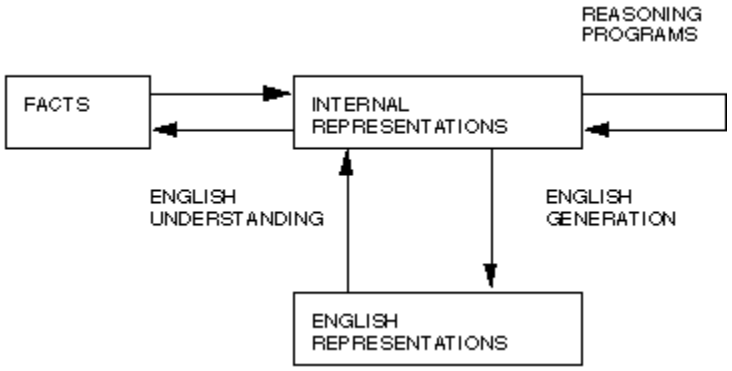


Fig: Two entities in Knowledge Representaion

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

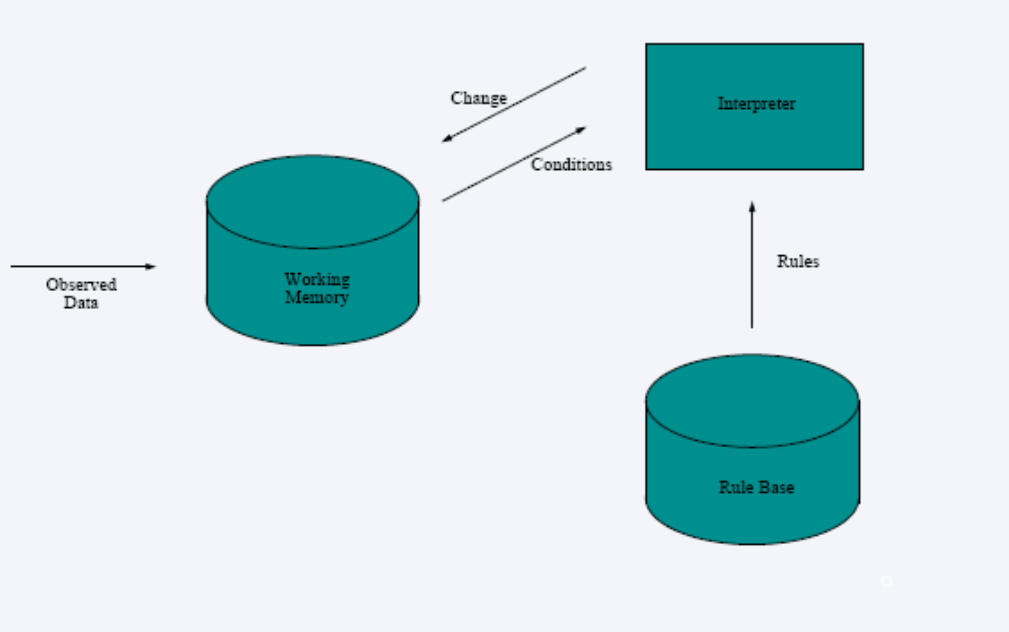
Notes:

- *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

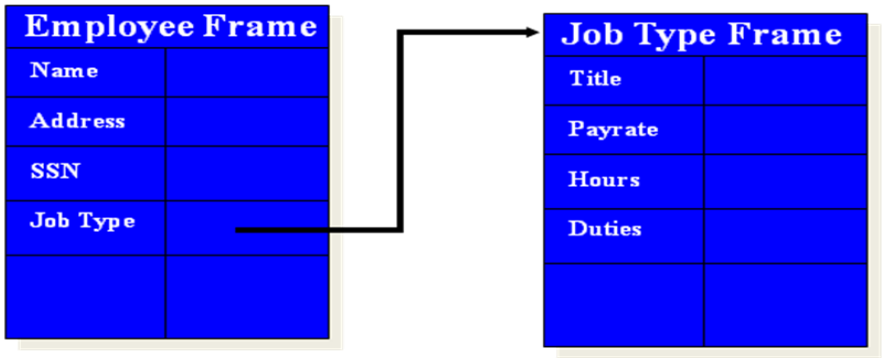
- complex
- reasoning (inferencing) is difficult
- explanation is difficult, expressive limitation

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.)

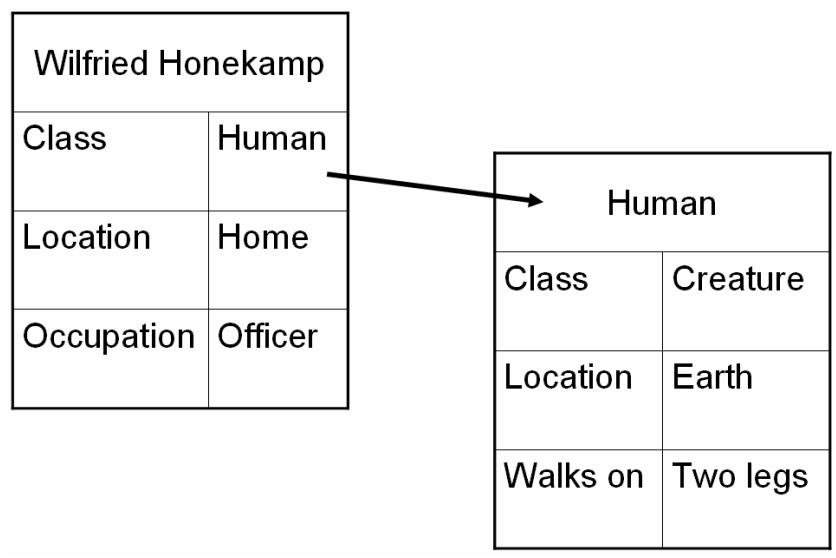


Fig: Two frames describing a human being

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***

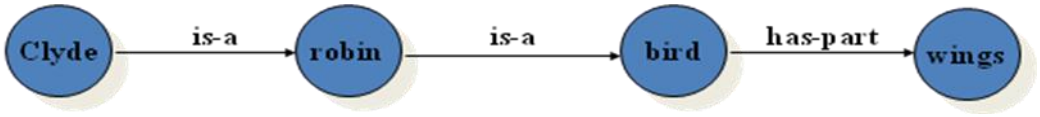
A node can represent a fact description
- physical object
- concept
- event

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:

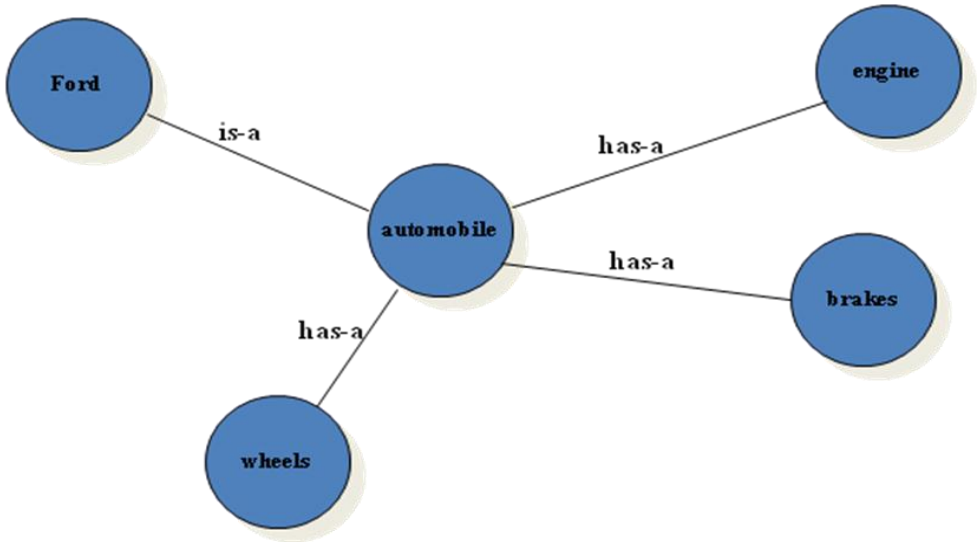


Fig: Automobile Semantic Net

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

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

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

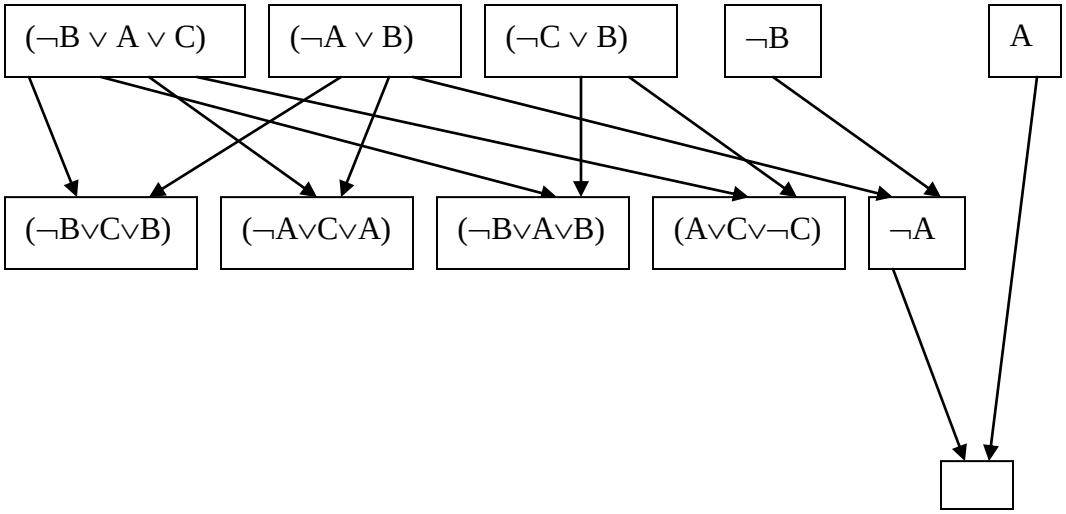
$(\neg B \vee A \vee C) \wedge ((\neg A \wedge \neg C) \vee B) \wedge \neg B$

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

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:

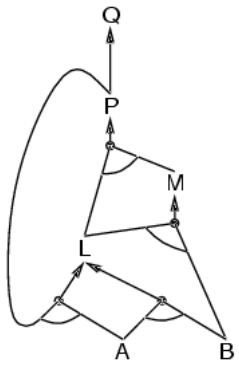
- ✓ Can be written as an implication
- ✓ Inference through forward chaining and backward chaining.
- ✓ Deciding entailment can be done in a time linear size of the knowledge base.

Forward chaining:

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

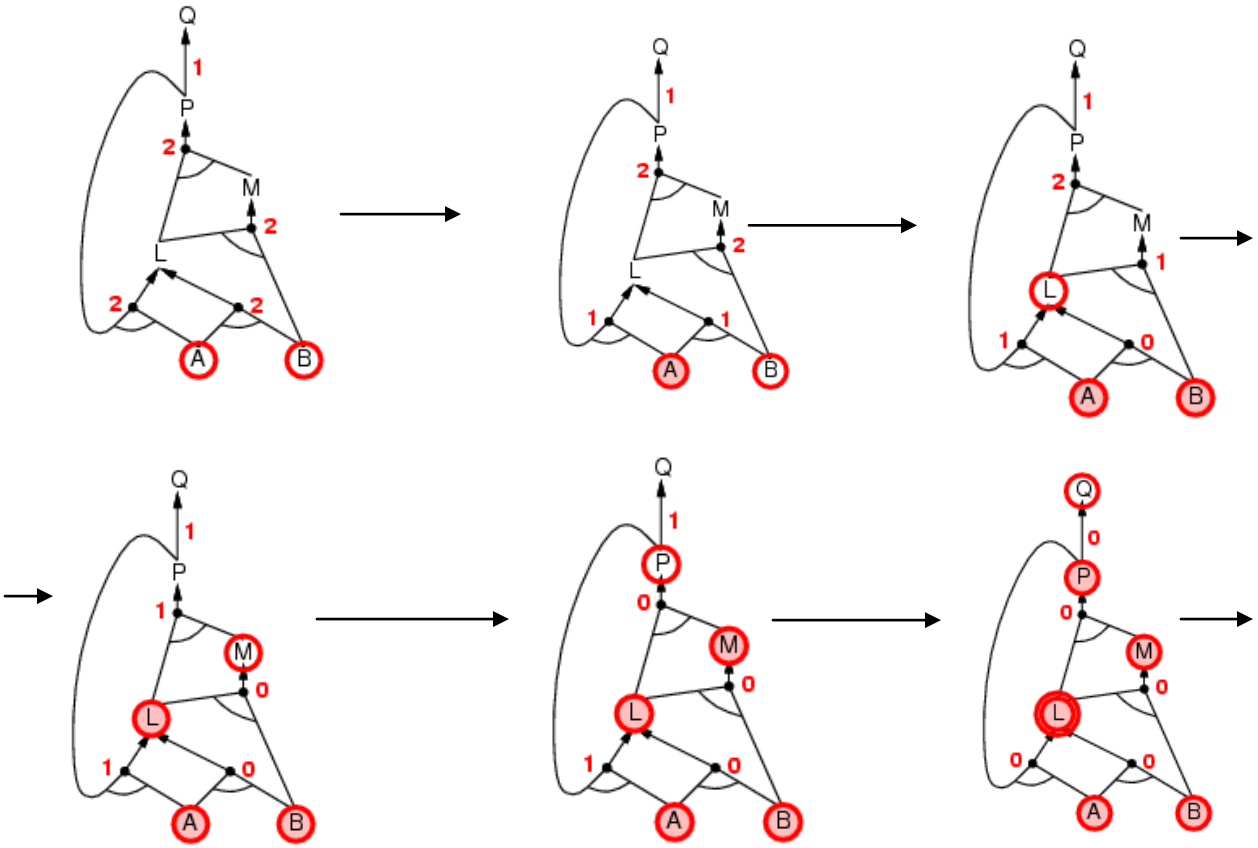
- add its conclusion to the *KB*, until query is found

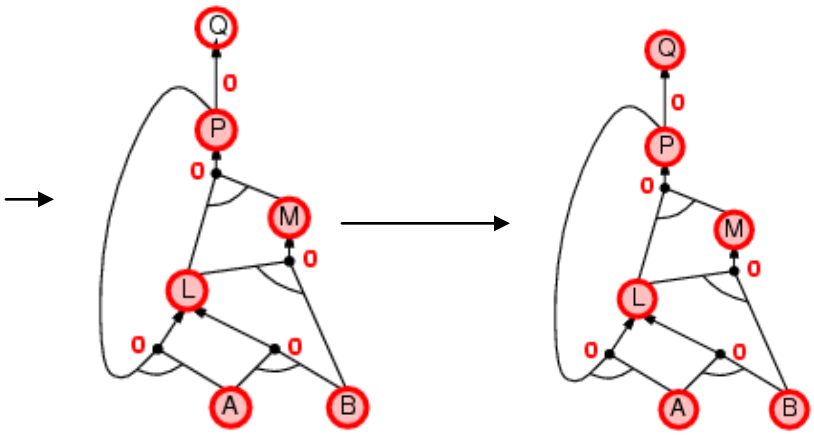
$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B



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*

$\text{At(KingJohn,UAB)} \wedge \text{Smart(KingJohn)} \vee \text{At(Richard,UAB)} \wedge \text{Smart(Richard)} \vee \text{At(UAB, UAB)} \wedge \text{Smart(UAB)} \vee \ldots$

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

existential quantifier $\exists x P(x)$ 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.

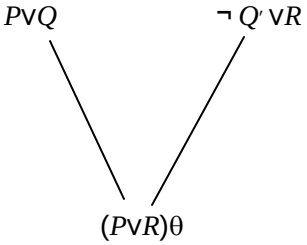
MGU = { y/John, x/z }

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.

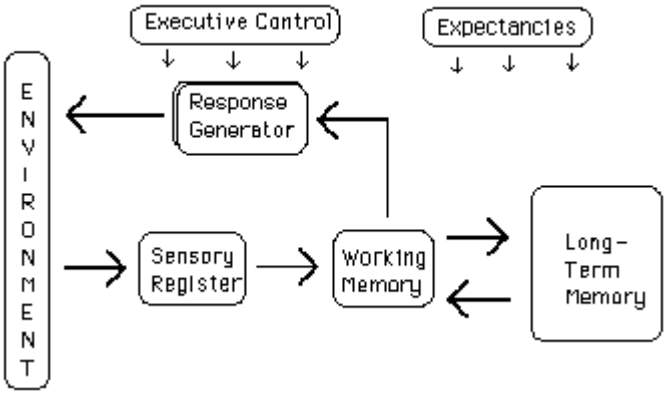


Fig: A model of human Information Processing

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.

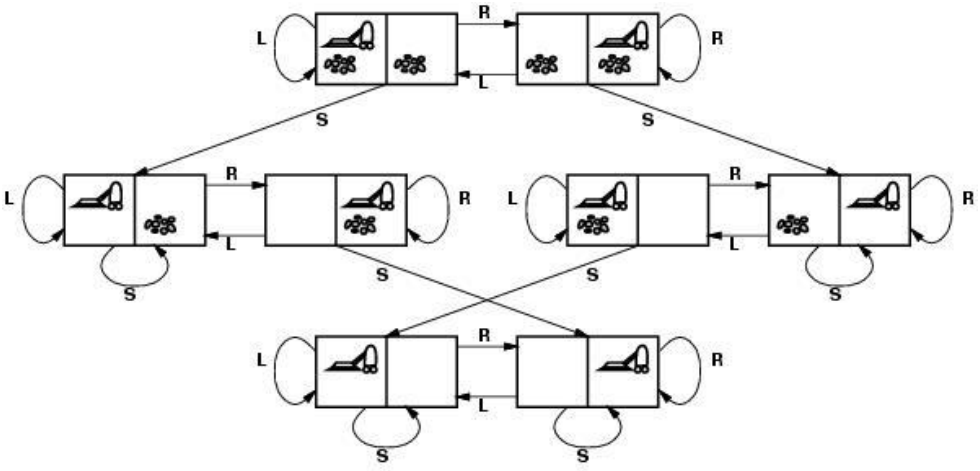
A solution is a sequence of actions from initial to goal state. Optimal solution has the lowest path cost.

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.

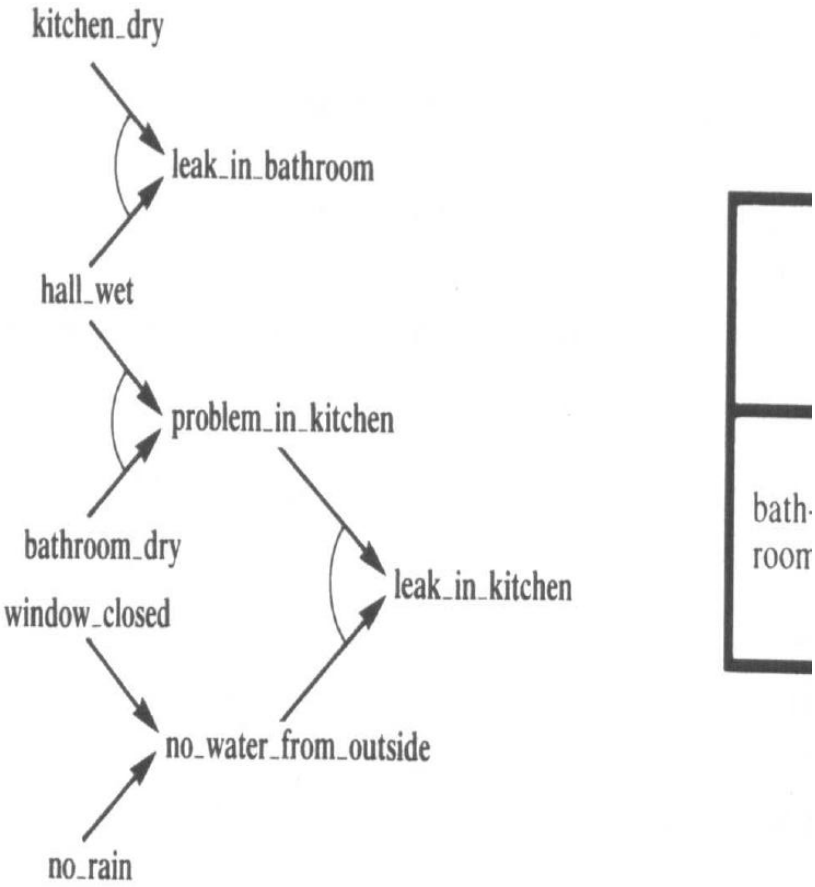
A **solution** is a path from the initial state to a goal state.

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

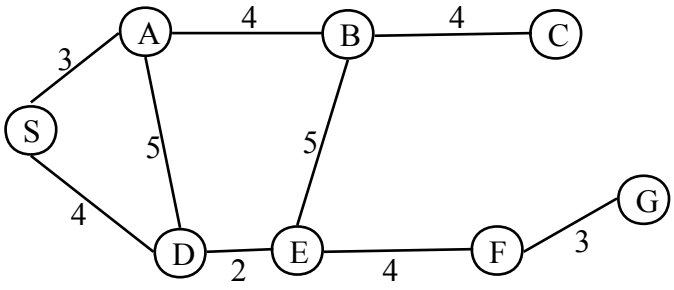


Figure : A graph representation of a map

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

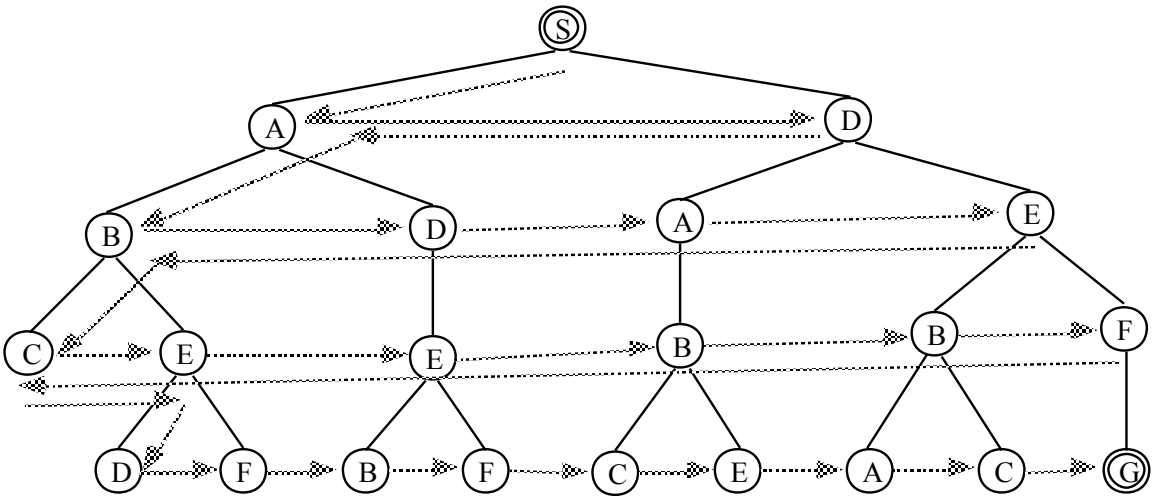
b -- maximum branching factor (number of successor of any node) of the search tree
d -- depth of the least-cost solution
m -- maximum length of any path in the space

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:

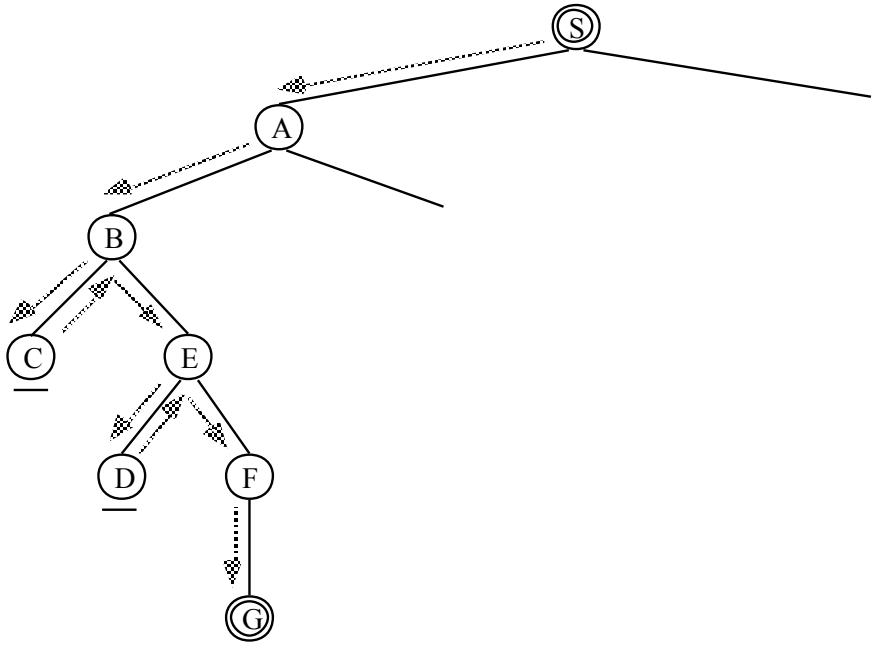
$$1 + b + b^2 + b^3 + \ldots\ldots\ldots\ldots\ldots\ldots b^d + (b^{d+1} - b) = \mathbf{O(b^{d+1})}$$

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

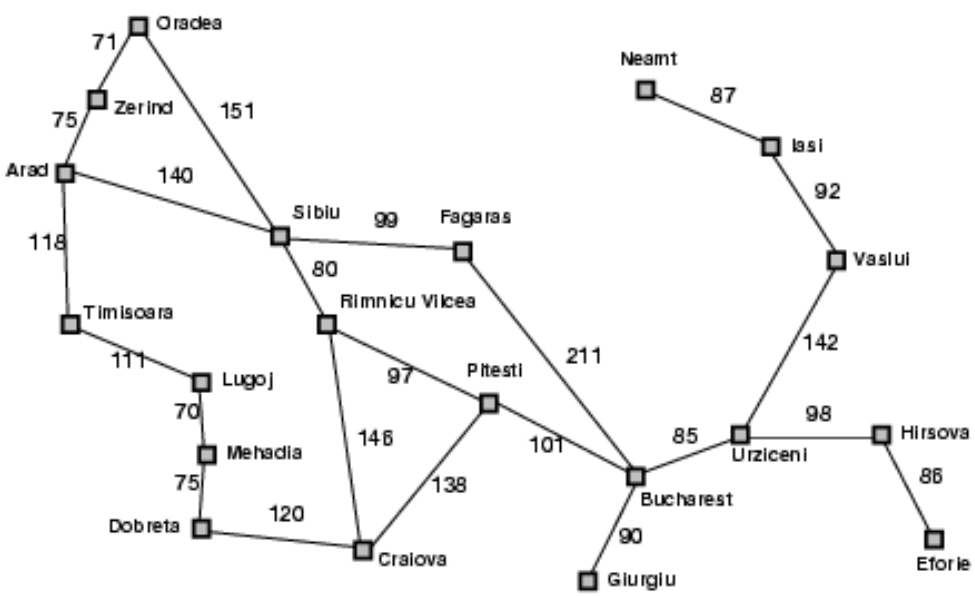
Greedy Best First Search

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

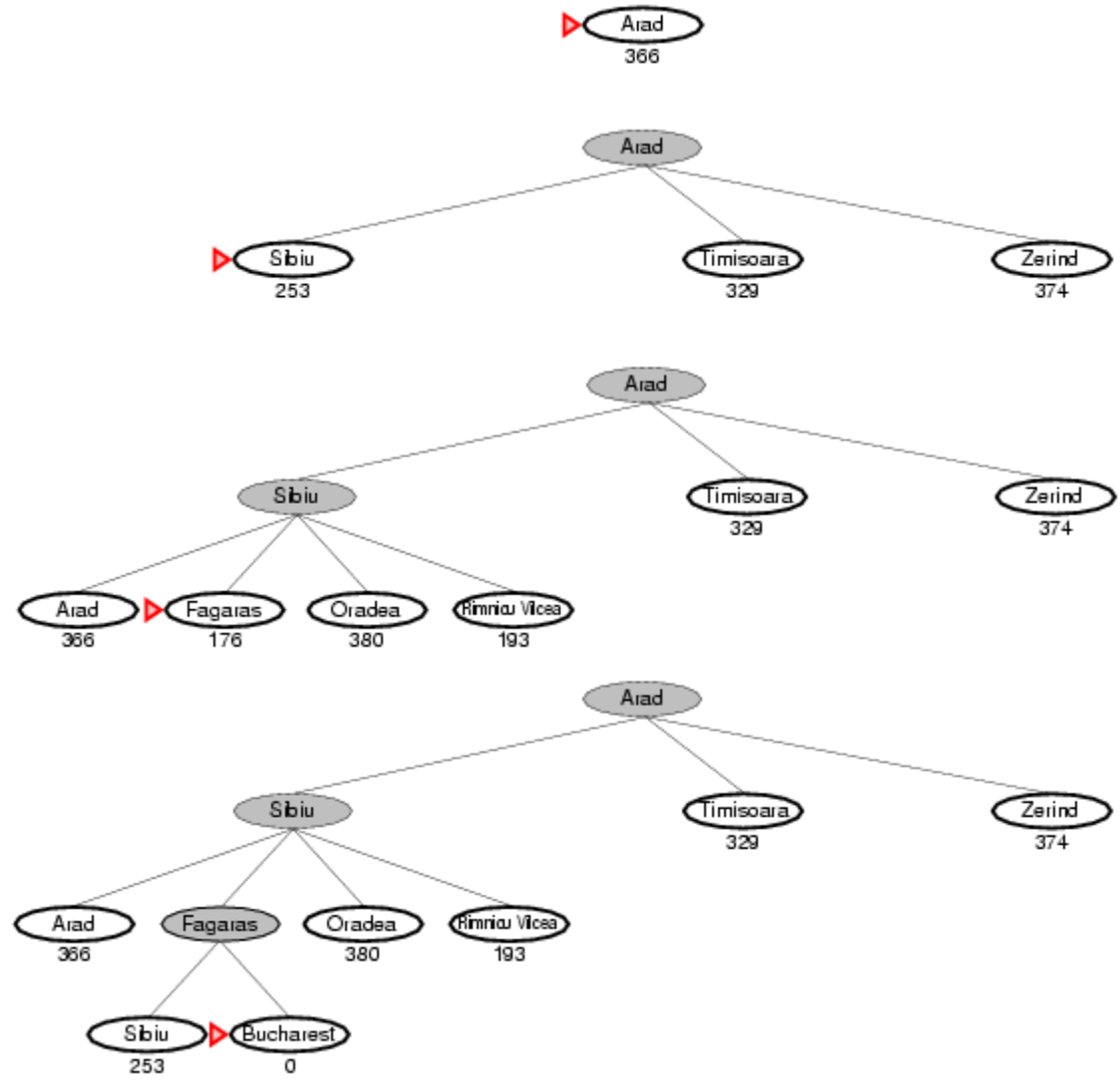
Greedy best-first search expands the node that appears to be closest to goal.

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.

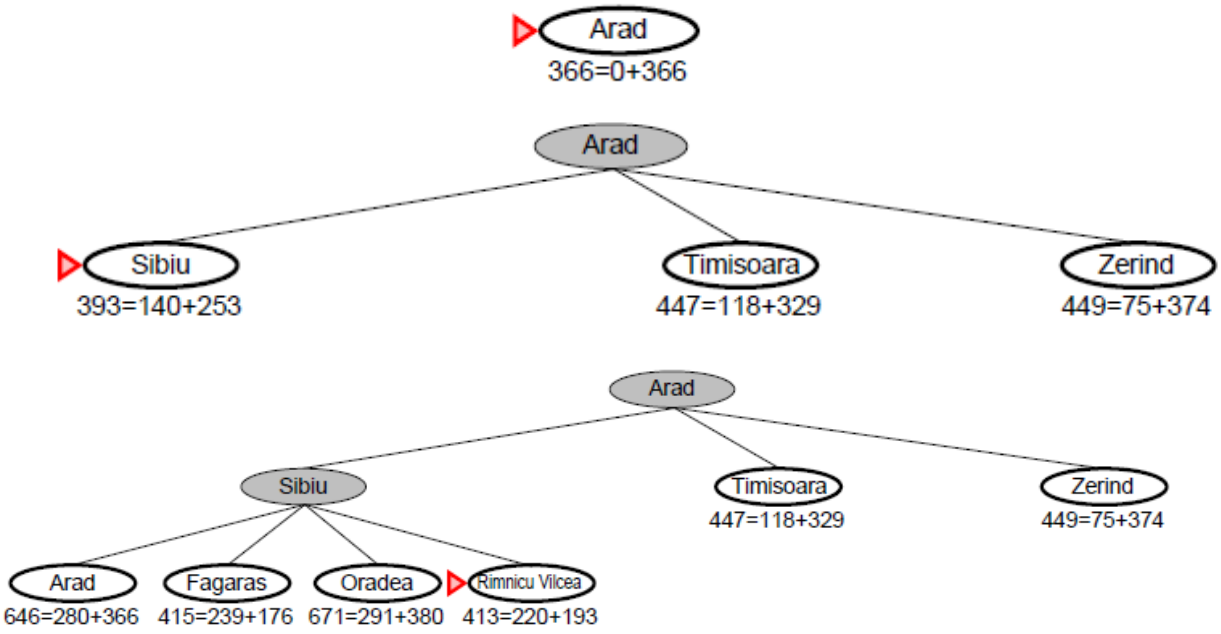
It finds a minimal cost-path joining the start node and a goal node for node n.

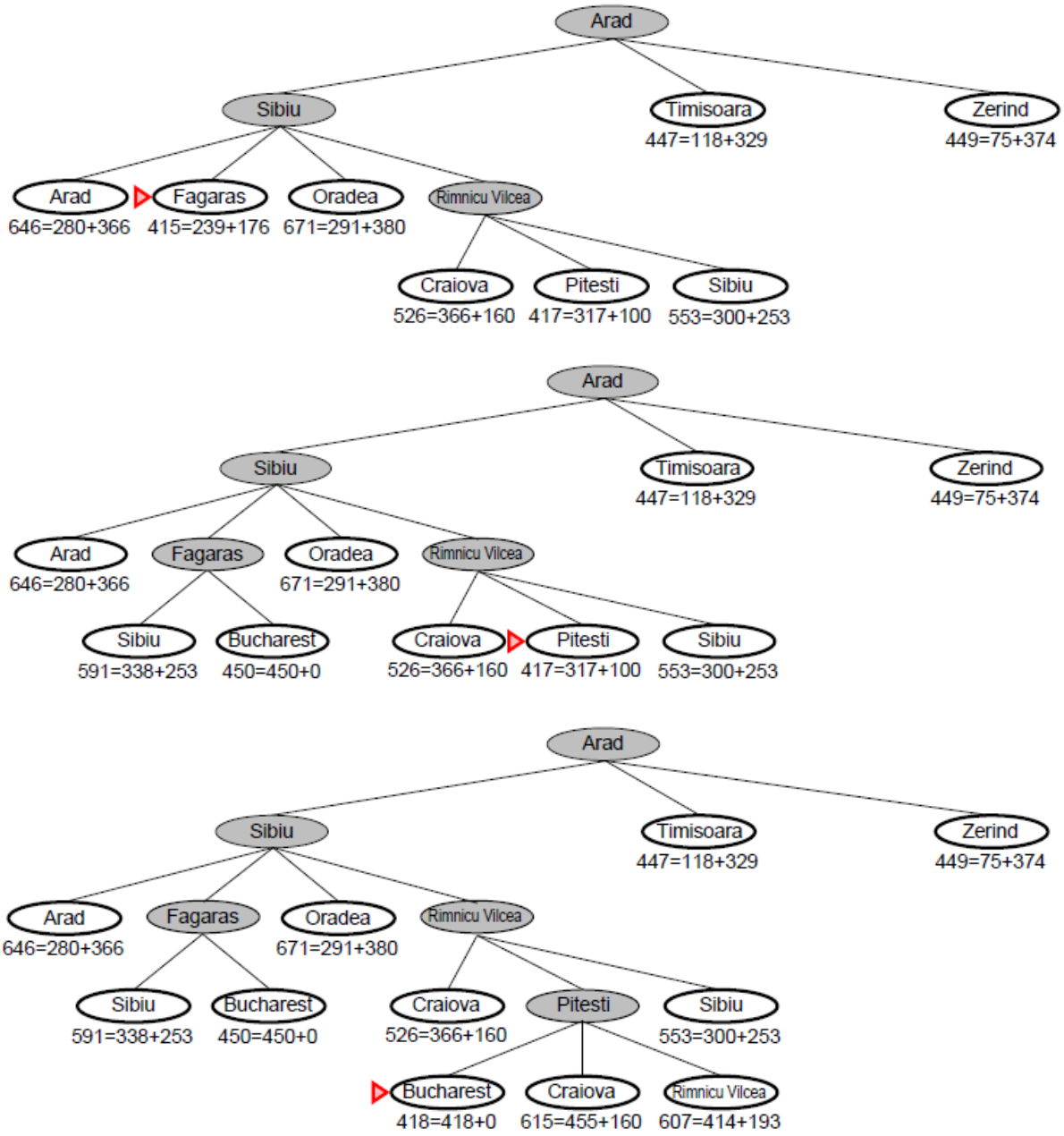
Evaluation function: f(n) = g(n) + h(n)

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.

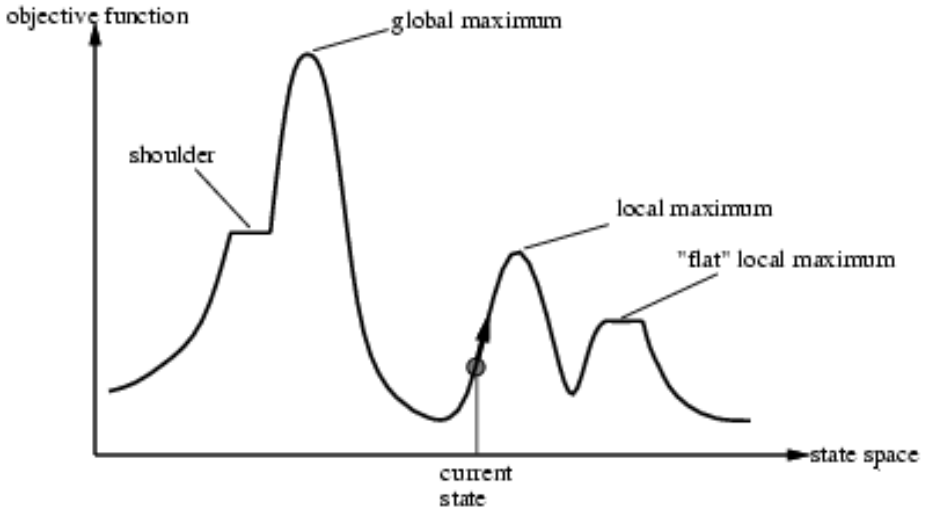
For example, hill climbing can be applied to the traveling salesman problem. It is easy to find a solution that visits all the cities but will be very poor compared to the optimal solution. The algorithm starts with such a solution and makes small improvements to it, such as switching the order in which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

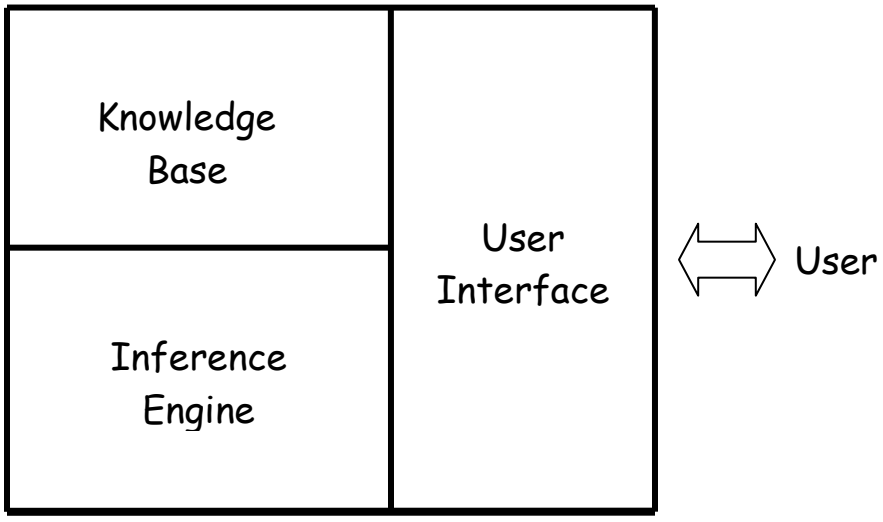


Fig: Block Diagram of expert system

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.

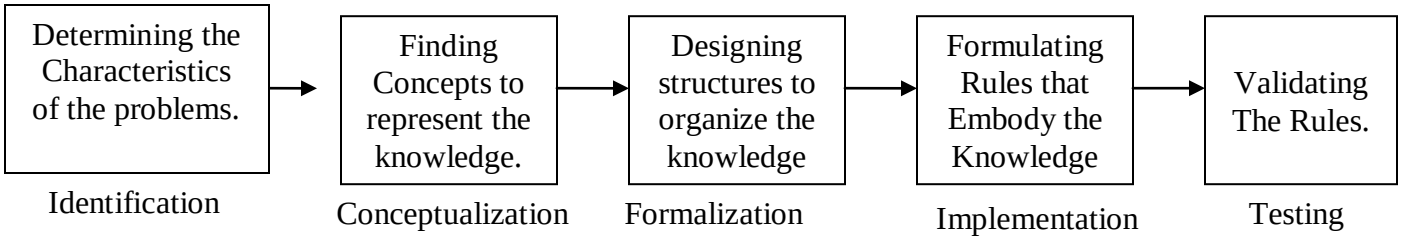


Fig: Different phases of expert system development

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:

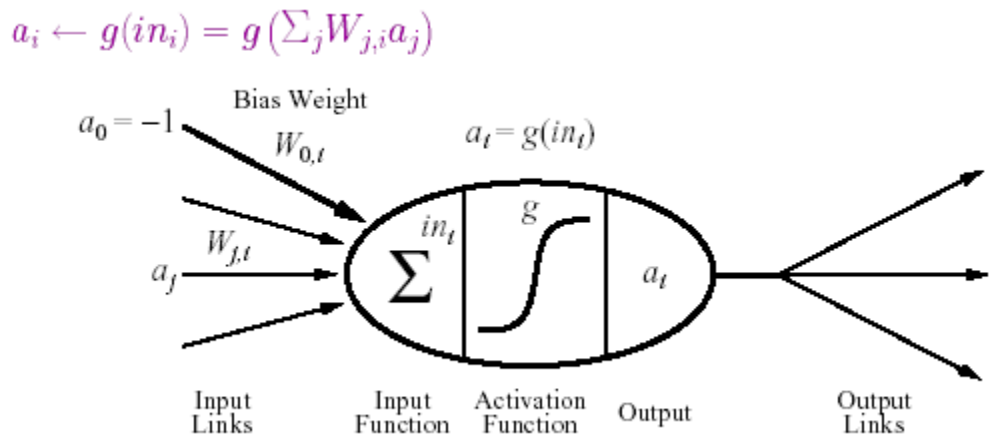
A function which is used to derive output activation from the input activations to a given node is called activation function.

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:

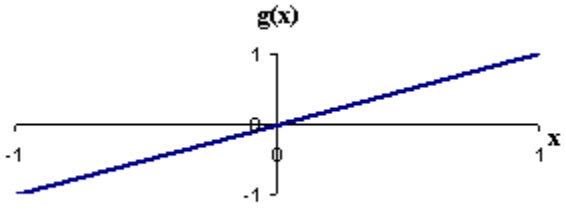
$$a_i = g(\ in_i) = g(\ \sum_{J=0}^{n} W_{j,i}\ a_j)$$

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



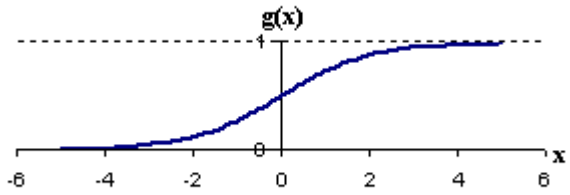
For **threshold activation functions**, the output are set at one of two levels, depending on whether the total input is greater than or less than some threshold value.

g(x) = 1 if x>= k

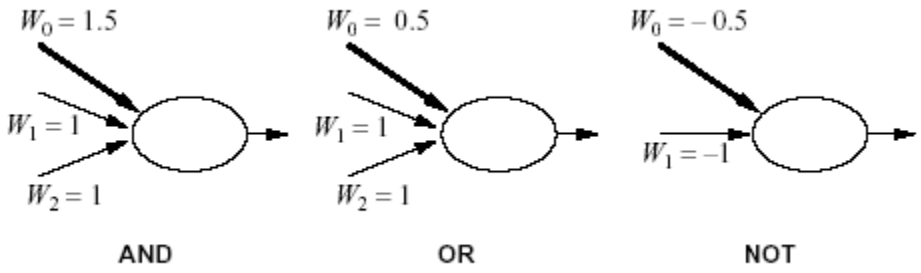
= 0 if x < k

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.

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



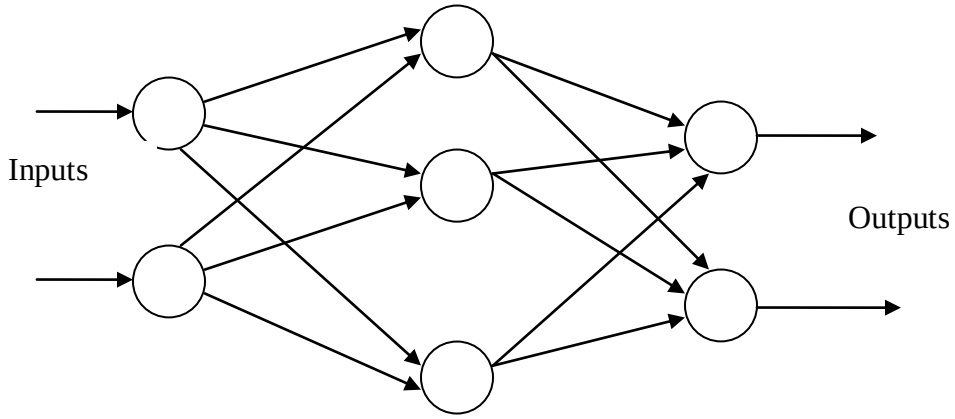
Realizing logic gates by using Neurons:



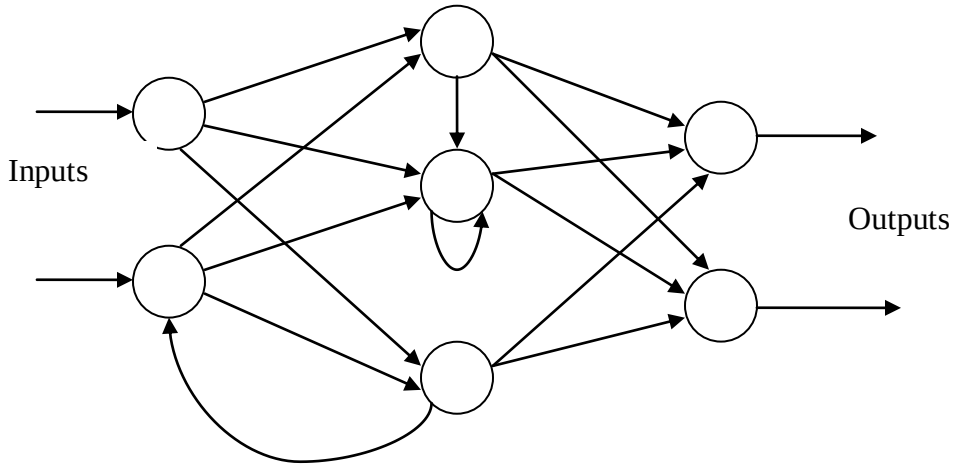
Network structures

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.

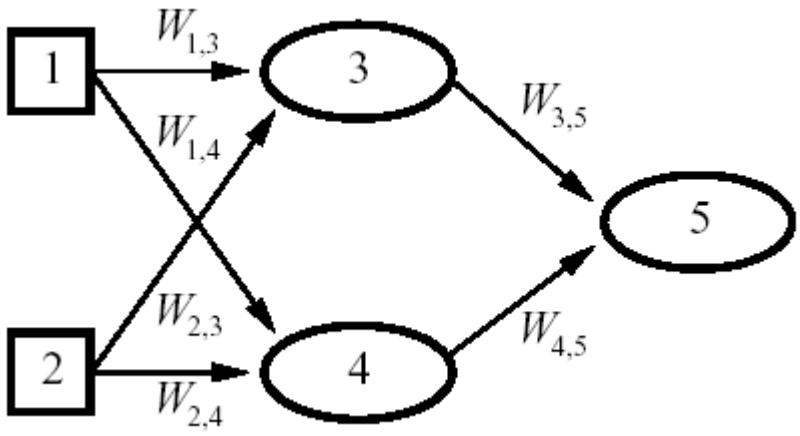


Feedback networks (Recurrent networks:)



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;

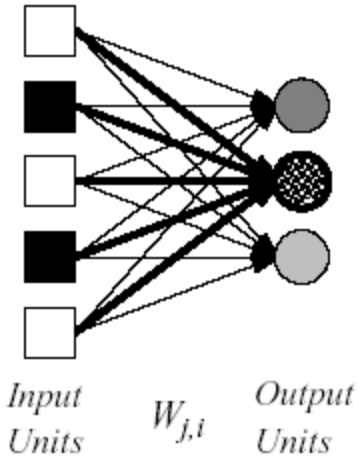
$a5 = g(W_{3;5}\ a_3 + W_{4;5}\ a_4)$

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

Types of Feed Forward Neural Network:

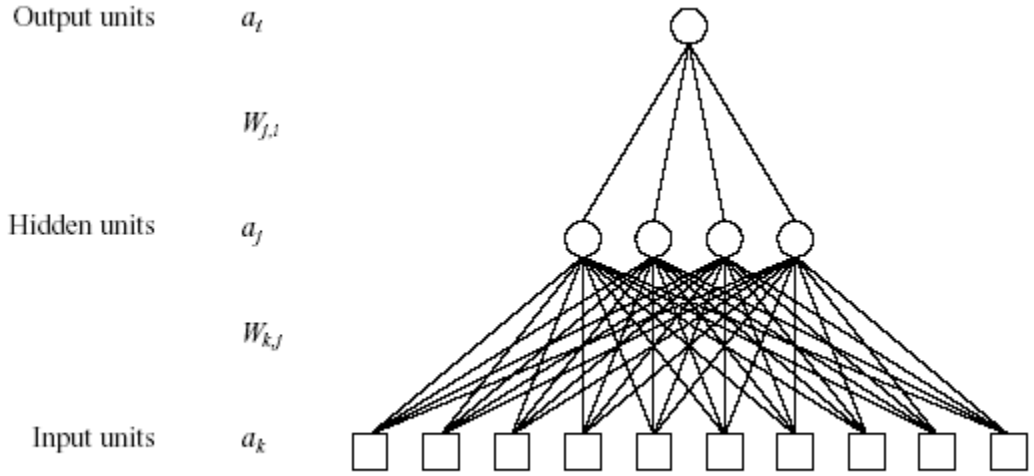
Single-layer neural networks (perceptrons)

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.