

HCS 404 ARTIFICIAL INTELLIGENCE (AI)

Lecturer: Mr. MUPFIGA (+263 772 373 525) OFFICE 31

COURSE OUTLINE:

1. Introduction to AI

Principles & Developments in AI

Brief History of AI

2. Intelligent Agents (Types of)

3. Problem Solving: Search Algorithms & heuristics

- Uninformed Search
- Informed Search

4. Knowledge and Reasoning

Agents that reason logically – propositional logic

- a) Using first-order logic
- b) Inference in first-order logic

5. Knowledge Representation

- Rules
- Frames
- Cases
- Semantic Nets

6. Rule Based Systems

- Forward Chaining
- Backward Chaining

7. Reasoning under Uncertainty

- Uncertainty
- Errors (types of)
- Uncertainty in Inference Chains

What is AI

Defn 1.

The science of making machines do things which would require intelligence if they were done by a human. (Marvin Minsky)

Defn 2.

(a) **A set of goals meant to** address a **(b) class of problems**, using (c) **a set of methods** by a **(d) set of people**

(a) A Set of Goals

- To make a machine that can do anything a human can do, i.e.
 - Understand natural language
 - Plan
 - Learn from experience

The aim is to build an artifact that is intelligent like a human being

(b) A Class of Problems

- Problems that require search:
 - No deterministic algorithm is known.
 - Must use “trial and error”.
 - Example: Schedule courses.
 - Non-example: Sort a class roster.

OR

- Problems that are poorly specified:
 - We don't know a concise, exact problem specification.
 - We don't know what knowledge is needed to solve the problem, and/ or we don't have the knowledge needed to solve the problem.
 - Our knowledge is imprecise or inaccurate.
 - Example: Explain integration to a human.
 - Non-example: Factor an integer.

(c) A Set of Methods

- Use of general inference methods
 - such as heuristic search, constraint propagation or resolution theorem proving.
- Representation of knowledge in declarative form,
 - such as search spaces, constraint networks

OR

A Set of Methods

- Heuristic Search
- Expert Protocols
- Iterative Programming
- More task-specific methods, e.g. for learning or planning
 - Tend to cross tasks
 - “AI-complete” problems

(d) A Set of People

X, Y, Z you & me

Artificial Intelligence can be defined in the form of:

- Human-centeredness
- Rationality
- Thought
- Action

-An aim to build systems that think and act like humans, and think and act rationally.

	Human	Rational
Think	*	*
Act	*	*

Acting Humanly: The Turing Test approach

- The Turing test: A human interrogates a subject through a tele-type.
 - If the human cannot tell whether the subject is a human or computer, the computer passes the test.
- To pass the Turing test a computer needs:
 - natural language processing
 - knowledge representation means
 - automated reasoning
 - machine learning

Thinking Humanly: The Cognitive Modeling approach

- Requires a model for human cognition, precise enough models allow simulation by computers.
 - Cognitive Science brings together Computer Science, Linguistics, Philosophy, and Psychology.

Thinking Rationally: The laws of thought approach

- Formal logic provides a precise notation for statements about things and their relationships.
 - Given sufficient memory and time, early computer programs were able to solve problems formulated in logical form, using automated reasoning and theorem proving techniques

But there is an obstacle,

- informal knowledge cannot be easily stated in logical notation.
- even small logic programs may have unacceptable time and memory requirements.

Acting Rationally: The rational agent approach

- means to act so as to achieve one's goals given one's beliefs.
 - an agent is something that perceives and acts.
 - may require thinking rationally to decide which action will achieve one's goal.
 - may require natural language, vision, and learning skills to be able to communicate

with the world and generate better strategies over time.

Advantages of the Rational Agent Approach:

- more general than the laws of thought approach.
- more amenable to study than the cognitive modeling approach.

Universal vs. Expert Abilities

- Abilities all normal adult humans have: (universal)
 - Seeing, hearing, walking, talking, learning, common sense.
- Abilities only some human experts have: (expert)
 - Proving theorems, playing chess, playing a musical instrument, managing companies, negotiating agreements.
- The expert abilities have turned out to be easier for AI machines.

Intelligent Agents

- An **agent** is an entity that perceives and acts. It:
 - Exists in an *environment*
 - Has *sensors* that detect *percepts* (*agent's perceptual input at any given time*) in the environment
 - Has *effectors* that can carry out *actions* which may change the environment
 - Has *goals* to achieve

Spectrum of Agent Complexity

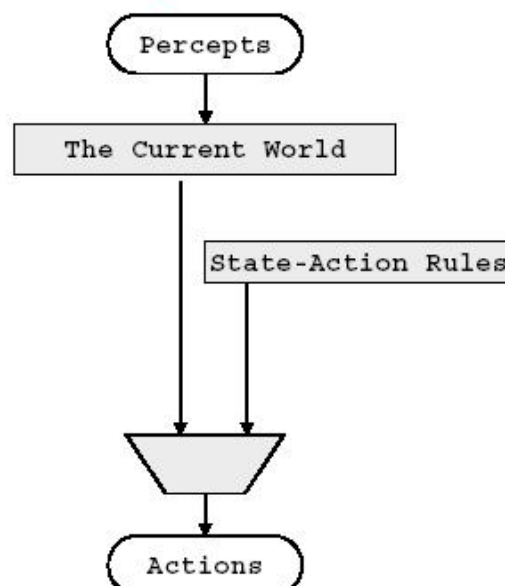
1. Simple reflex agents

Percepts -> actions, i.e. their action is based on the current percept, of limited intelligence.

1. Reflex Agent

Classes of agent:

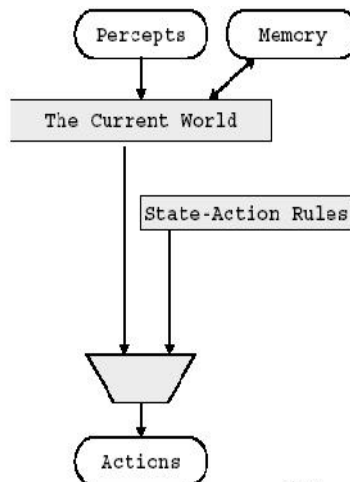
- Reflex
- Reflex with State
- Planning with Goals
- Planning with Utility



2. With internal state (Model-Based)

- Have memory of past percepts & actions. Model – knowledge about how the world works. E.g. finite state machine, vending machine

- Reflex
- Reflex with State
- Planning with Goals
- Planning with Utility



CS520: Steinberg

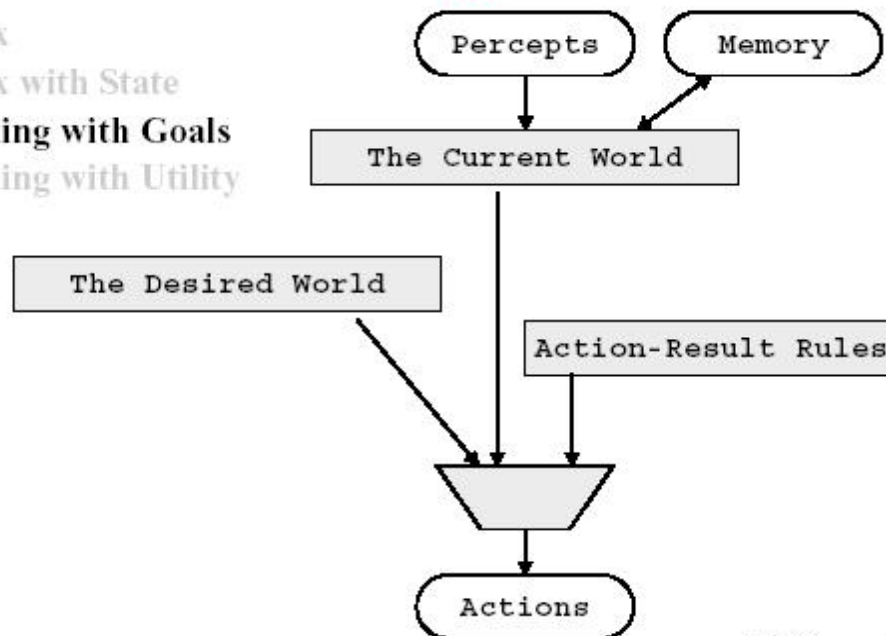
Lecture 02 5

3. Goal Based

- Explicit representation of desired state(s) of environment, more flexible & knowledge can be modified.

Classes of agent:

- Reflex
- Reflex with State
- Planning with Goals
- Planning with Utility



20: Steinberg

Lecture 02

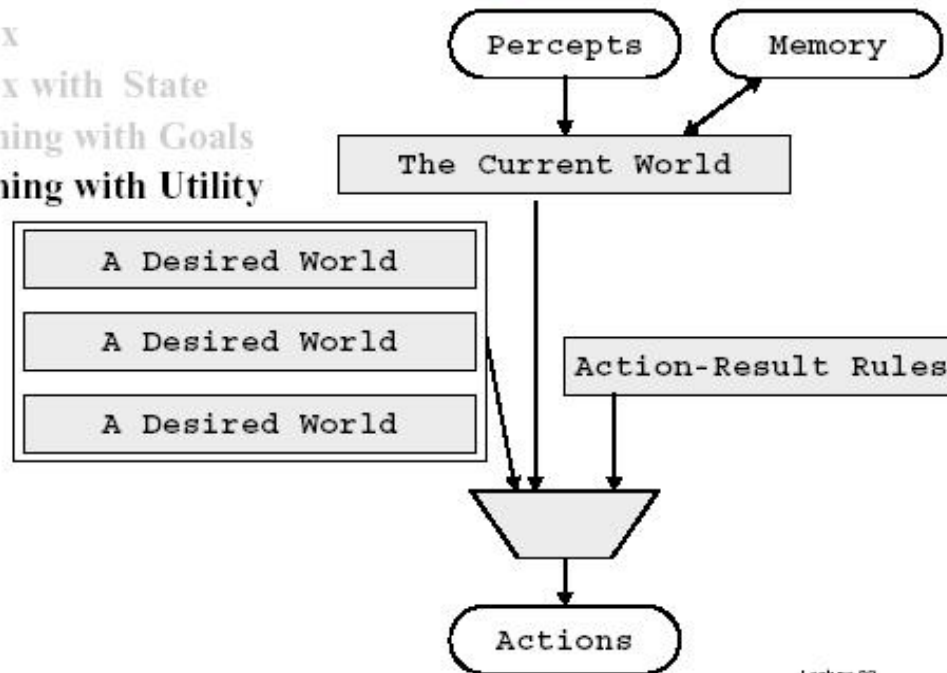
6

- Utility-based

- *How much* is each goal desired. A utility function maps a state onto a real number, which describes the associated degree of happiness. A complete specification of the utility function allows rational decisions in two kinds of cases, where goals are inadequate. First, when they are conflicting goals, the function specifies the appropriate tradeoff. Second, when there are several goals that the agent can aim for, none of which can be achieved with certainty, utility provides a way in which the likelihood of success can be weighed up against the importance of the goals.

Classes of agent:

- Reflex
- Reflex with State
- Planning with Goals
- Planning with Utility



5.20: Steinberg

Lecture 02

Environments

- accessible vs. inaccessible (complete state of environment accessible to the agent at each point)
- deterministic vs. non-deterministic (stochastic), next state based on current state.
- episodic vs. non-episodic (each episode consists of the agent perceiving & then performing a single independent action)
- static vs. dynamic (environment changes while agent performs action)
- discrete vs. continuous (the way time is handled, & to the percepts & actions of the agent).

History of Artificial Intelligence

A. The Gestation of AI (1943 – 1955)

- Model of artificial neurons
- The use of simple updating rule for modifying the connection strengths between neurons.

B. The Birth of AI (1956)

- Dartmouth conference by John McCarthy
- Presentation of the Logic Theorist (PRG for basic logic theorem proving) by Newell & Simon

C. AI Early Work (1952 – 1969)

- Newell & Simon's General Problem Solver
- Gelerter(1959) Geometry Theorem Prover
- Definition of LISP by McCarthy
- Development of microworlds programs by Minsky's students

D. Discovery of Reality(1966 – 1973)

- Identification of perceptrons limitations in terms of knowledge representation.

E. Knowledge-Based Systems(1969-1979)

- Introduction to the use of domain specific knowledge to solve complex prob.
- Design of DENDRAL used to solve the problem of inferring molecular structure from info. Provided by a mass spectrometer
- Design of MYCIN used to diagnose blood infections.
- Birth of PROLOG

F. AI Industry Boom (1980 - 1990)

Commercial expert system (R1) used to configure orders for new computer systems at DEC.

US sales of AI-related hardware and software soared to \$425 million in 1986.

G. The Return of neural networks(1986 – 1990)

- Hopfield neural networks developed

H. AI becomes a science(1987 – Now)

- Experiments on Speech recognition & neural networks carried out
- Introduction of data mining technology
- Bayesian Networks used to reason with uncertain knowledge
- Rationally acting expert systems designed

I. Beginning of Intelligent Agents (1995 – Now)

Application Areas of Artificial Intelligence

1. Knowledge Based Systems (**will focus on this in this course**)

2. Natural Language Processing

One goal of AI work in natural language is to enable communication between people and computers without resorting to memorization of complex commands and procedures. Automatic translation---enabling scientists, business people and just plain folks to interact easily with people around the world---is another goal. Both are just part of the broad field of AI and natural language, along with the cognitive science aspect of using computers to study how humans understand language.

Natural language generation systems convert information from computer databases into normal-sounding human language, and natural language understanding systems convert samples of human language into more formal representations that are easier for computer programs to manipulate.

Would be examples of NLP applications:

- A computer answering the phone, and replying to a question
 - understanding the text on a Web page to decide who it might be of interest to
 - translating a daily newspaper from Japanese to English
 - understanding text in journals / books and building an expert system based on that understanding
3. Pattern Recognition (uses shapes, forms, outlines, color, surface texture, temperature, or other attribute). Pattern recognition aims to classify data (patterns) based on either a priori knowledge or on statistical information extracted from the patterns. The patterns to be classified are usually groups of measurements or observations, defining points in an appropriate multidimensional space.
 4. Intelligent Computer Assisted Learning
An adaptive system that is able to modify its teaching strategy to suit the student being taught, i.e.
 - Must be able to model the pattern of the student's responses & adapt in a way
 - Must be able to reason about the knowledge it has regarding the student being taught
 - Allows for flexibility in the interaction between student & system
 5. Speech Recognition
 6. Models of Human Cognition

Lecture 3

Problem Solving by Search

Problem Solving Agents are Goal Based agents, which decide what to do by finding sequences of actions that lead to desirable states. A problem in the state-space search is defined as follows (S O G) where

S is the set of one/more initial states

O is the set of operators on states

G is a set of goal states

The state space is identified by a directed graph in which each node is a state and each arc represents the application of an operator transforming a state onto a successor state. A solution is a path from a start state to a goal state. A goal state may be defined either explicitly or as the set of states satisfying a given predicate

Goal Formulation

A goal is a set of world states in which a goal is satisfied. Goal formulation is the first step in problem solving. The agent's task is to find out which sequence of actions will get it to a goal state.

Problem formulation is the process of deciding what actions and states to consider, given a goal. Search is the process of looking for possible sequences of actions that lead to states of known value, and choosing the best sequence when they are several options.

A **search algorithm** takes a problem as input and returns a solution in the form of an action sequence.

Example Agent Design (Oversimplified example)

Function SIMPLE-PROBLEM-SOLVING AGENT (percept) returns an action

input: percept, a percept

static: seq, an action sequence, initially empty

state, some description of the current world

goal, a goal, initially null

problem: a problem formulation

state \leftarrow UPDATE-STATE (state, percept)

if seq is empty then do

goal \leftarrow FORMULATE-GOAL(state, percept)

problem \leftarrow FORMULATE-PROBLEM(state, goal)

seq \leftarrow SEARCH(problem)

action \leftarrow FIRST(seq)

seq \leftarrow REST (seq)

return action

Here the environment is assumed to be **static** and can be viewed as **discrete**, and the initial state is known. The environment is also assumed to be **deterministic**. This is an oversimplified case.

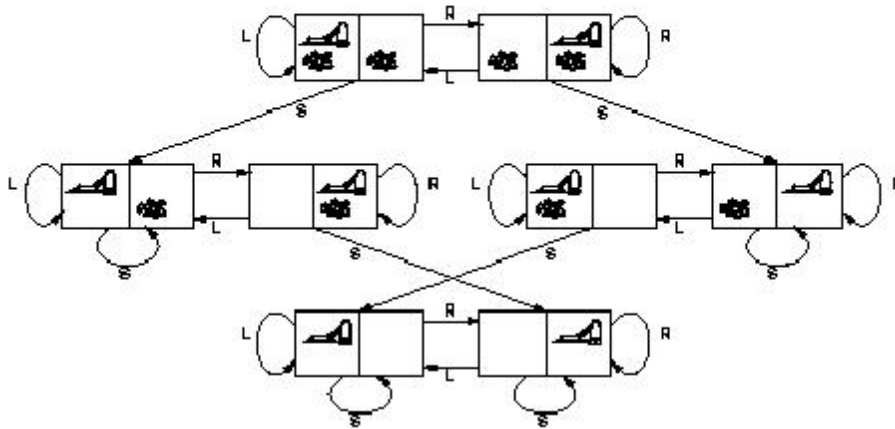
Formally a problem can be defined as having four components:

- An initial state, the agent starts in
 - A set of states (the state space, i.e. the initial state & a set of all possible successor states (successor function)).
 - A path cost function assigning a numeric cost to each path.
- A goal test, which determines whether a given state is a goal state.

Example Problems (Toy Problems)

1. The vacuum world (with a sensor-less agent) Search space

The vacuum world



- Problem formulation as search:
 - **states**: one of the 8 shown above.
 - **operators**: move left, move right, or suck.
 - **goal test**: no dirt left in any square.
 - **path cost**: each action costs 1; path cost=path length.

2. The 8-puzzle

5	4	
6	1	8
7	3	2

Start State

1	2	3
8		4
7	6	5

Goal State

- Problem formulation as search:
 - states: location of each tile as well as blank tile.
 - operators: blank moves left, right, up, or down.
 - goal test: state matches goal state shown.
 - path cost: each move costs 1; path cost=path length.

Question: Show the state space for the 8 puzzle???

Some Search Complications

- Multiple paths
- Cycles
- Infinite paths

Search Method Evaluation

Criteria • **Correctness:** Are all "solutions" found by the algorithm correct?

- **Completeness:** Does the algorithm find a solution, whenever one exists?
- **Termination:** Is the algorithm guaranteed to terminate on all problems?
- **Solution Optimality:** Is the algorithm guaranteed to find optimal solutions?
- **Complexity:** How much time and space does the algorithm use?
- **Algorithm Optimality:** Does the algorithm use as little time or space as possible?

Blind Search (Uninformed) Search Algorithms

1. Breadth First Search
2. Uniform Cost search
3. Depth First Search
4. Depth Limited
5. Iterative Deepening
6. Bi-directional Search

Lecture 4

Informed Search Strategies

- The search space of a problem is generally described using the number of possible states in each search.
- Some problems are too large to search efficiently using uniformed methods.
- At times we have additional knowledge about the problem that we can use to inform the agent doing the search.
- Heuristics (informed guesses) are employed to direct the search.
- Heuristics can estimate the goodness of a particular node (or state) n . i.e.
 - how close is n to a goal node.
 - What might be the minimal cost path from n to a goal node?
- Formally, $h(n) \geq 0$ for all nodes
- $h(n) = 0$ means n is a goal node.
- $h(n) = \infty$ implies n is a dead end, does not reach a goal state.
- e.g. heuristic $h_{SLD}(n)$ = straight line distance from n to goal.

A/ Best-first search

- General approach of informed search:
- Node is selected for expansion based on an *evaluation function* $f(n)$
- Evaluation function measures distance to the goal.
- Choose node, which *appears* best
- It can be implemented as:

function BEST-FIRST-SEARCH(*problem*, EVAL-FN) **returns** a solution sequence
inputs: *problem*, a problem
Eval-Fn, an evaluation function
Queueing-Fn \leftarrow a function that orders nodes by EVAL-FN
return GENERAL-SEARCH(*problem*, *Queueing-Fn*)

Has variations, that use some estimated measure of the cost of the solution and try to minimize it, e.g. cost of the path so far from the initial state to the present state. In order to focus the search, the measure must incorporate some estimate of the cost of the path from a particular state to the goal.

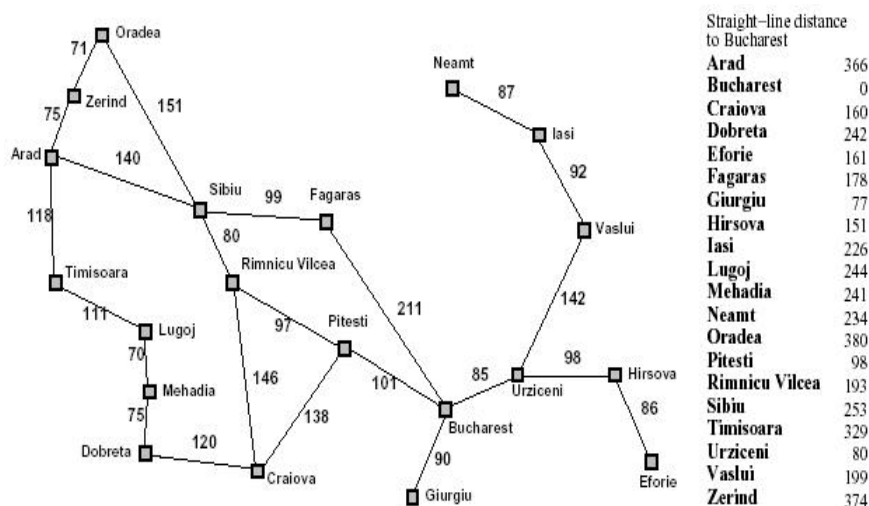
i) Greedy Best-Search Algorithm: $f(n) = h(n)$

-A form of Best First-Search

-Expands the node closest to the goal, on the grounds that this is likely to lead to a solution quickly.

-Uses a heuristic function: $f(n) = h(n)$ where $h(n)$ is the cost of moving from the current node to the goal.

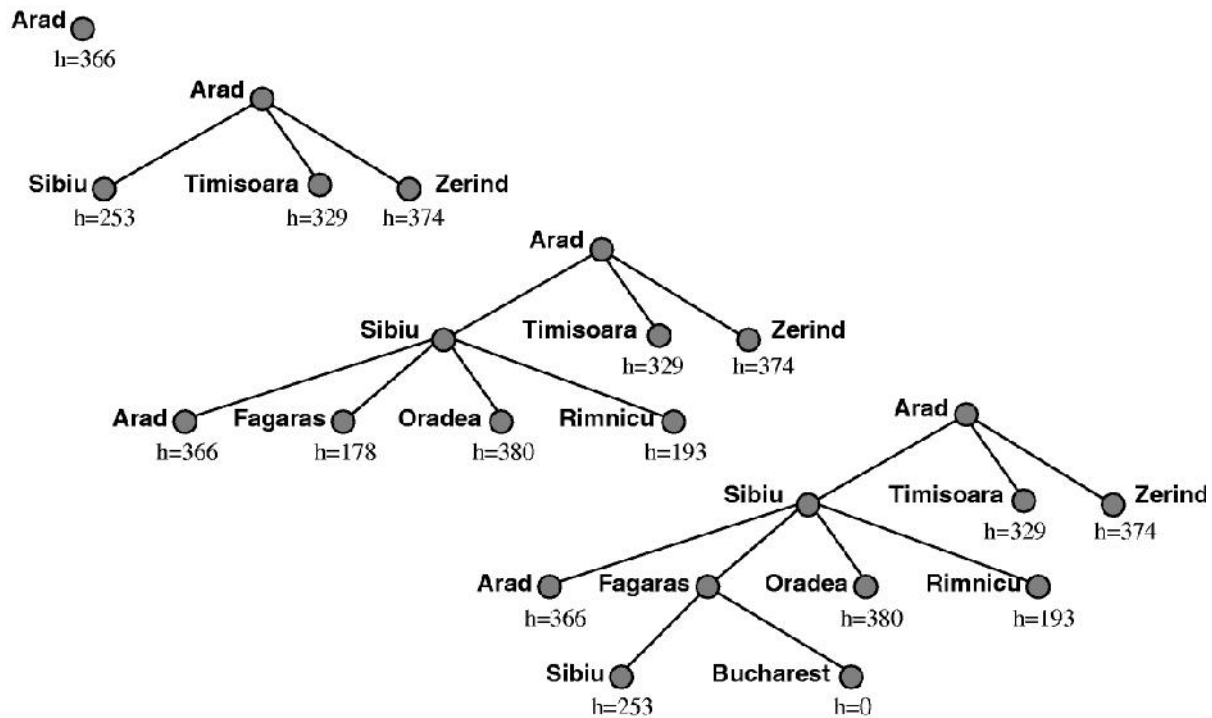
Suppose the following map with distances between towns as shown.



Assume the goal is to move from Arad to Bucharest.

Using Greedy Best Search First, the path with the immediate shortest route to Bucharest is followed.

-Suppose the function $f(n) = h_{SLD}(n)$ is employed using the Greedy Best search algorithm where SLD is straight line distance from node n to Bucharest as given in the map. The figure below shows how to get to Bucharest using Greedy Best Search Algorithm with the prescribed function.



The algorithm finds a path, though not an optimal path: the path it found via Sibiu and Fagaras to Bucharest is 32 miles longer than the path through Pimnicu Vilcea and Pitesti. Hence, the algorithm always chooses what looks locally best, rather than worrying about whether or not it will be best in the long run.

Greedy search is susceptible to false starts. Consider the problem of getting from Iasi to Fagaras. h suggests that Neamt be expanded first, but it is a deadend. The solution is to go first to Vaslui and then continue to Urziceni, Bucharest and Fagaras.

NB, if we are not careful to detect repeated states, the solution will never be found - the search will oscillate between Neamt and Iasi.

Greedy search resembles depth first search (dfs) in the way that it prefers to follow a single path to the goal and backup only when a dead-end is encountered. It suffers from the same defects as dfs - it is not optimal and it is incomplete because it can start down an infinite path and never try other possibilities.

The worst-case complexity for greedy search is $O(b^m)$, where m is the maximum depth of the search. Its space complexity is the same as its time complexity, but the worst case can be substantially reduced with a good heuristic function.

ii) A* search

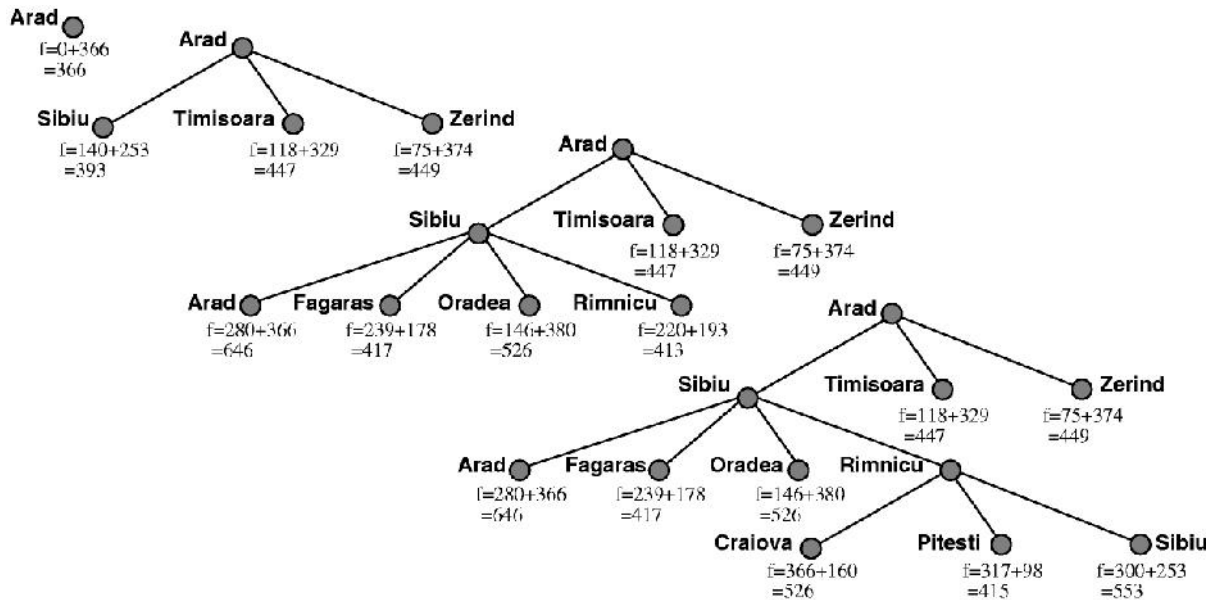
-Combines the use of $g(n)$, the cost of the path so far, and $h(n)$, the cost of moving from the node n to the goal, simply by summing them: $f(n)=g(n)+h(n)$

- $f(n)$ = the estimated cost of the cheapest solution through node n .

-Minimizes the total estimated cost, and is optimal if $h(n)$ is an admissible heuristic(i.e. no overestimation of reaching goal).

-Avoid expanding paths that are already expensive (consider the lowest $g(n)$).

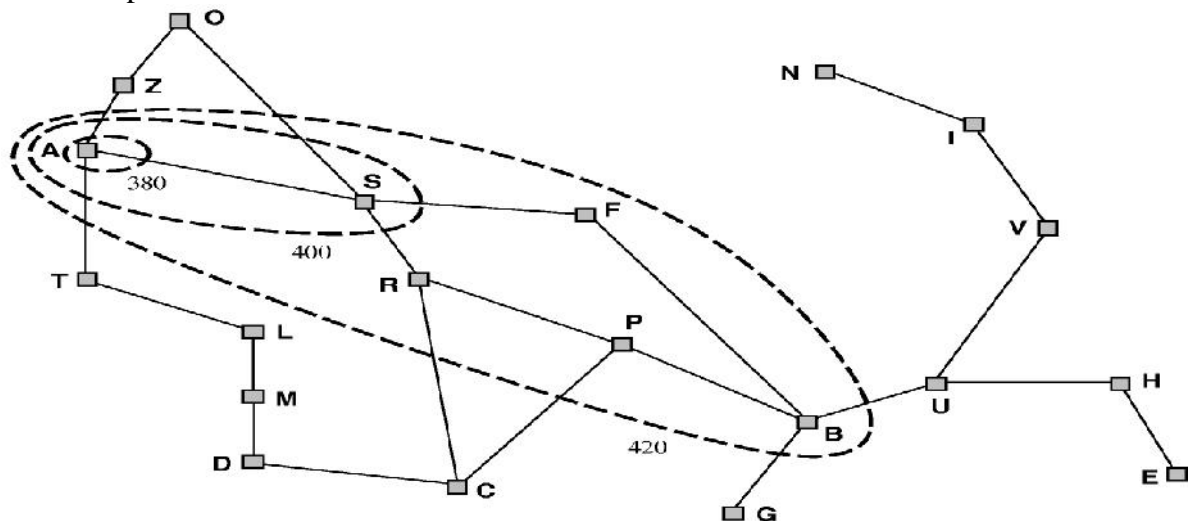
The following is a search from Arad to Bucharest using A* Search



Notice that A^* prefers to expand from Rimnicu Vilcea rather than Fagaras. Even though Fagaras is closer to Bucharest, the path taken to get to Fagaras is not as *efficient* in getting close to Bucharest as the path taken to get to Rimnicu.

For the A^* search above: along any path from the root, the f-cost never decreases. This fact holds true for almost all admissible heuristics. A heuristic with this property is said to exhibit **monotonicity**.

If f never decreases along any path out of the root, we can conceptually draw contours in the state space.



Because A^* expands the leaf node of lowest f, an A^* search fans out from the start node, adding nodes in concentric bands of increasing f-cost.

In A^* , the first solution found must be the optimal one because nodes in all subsequent contours will have higher f -cost and hence higher g -cost.

A^* search is also complete because as we add bands of increasing f , we must eventually reach a band where f is equal to the cost of the path to a goal state.

A^* is optimally efficient for any given admissible heuristic function.

Complexity of A^*

The catch with A^* is that even though its complete, optimal and optimally efficient, it still can't always be used, because for most problems, the number of nodes within the goal contour search space is still exponential in the length of the solution.

Similarly to breadth-first search, however, the major difficulty with A^* is the amount of space that it uses.

Recursive Best-First Search (a variation of A^*)

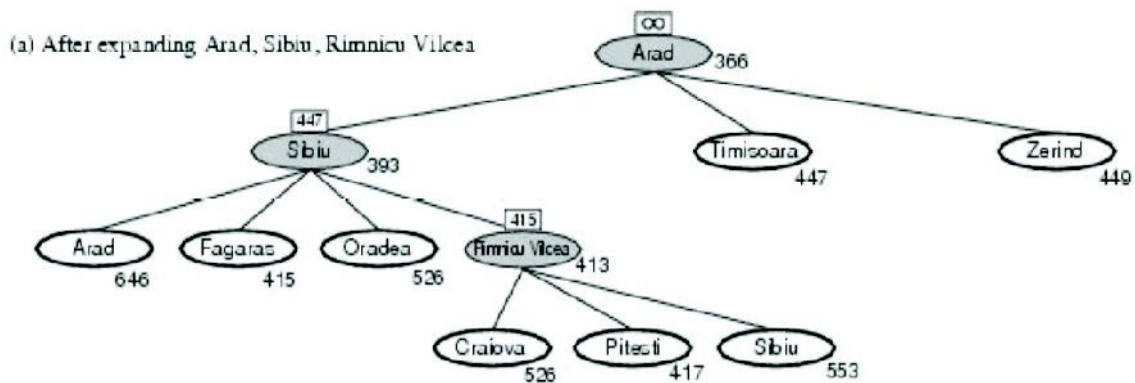
- A^* wastes a lot of memory, just like breadth first

-**RBS** a variation of A^* with backtracking, attempts to mimic the operation of a best-first search algorithm, using only linear space.

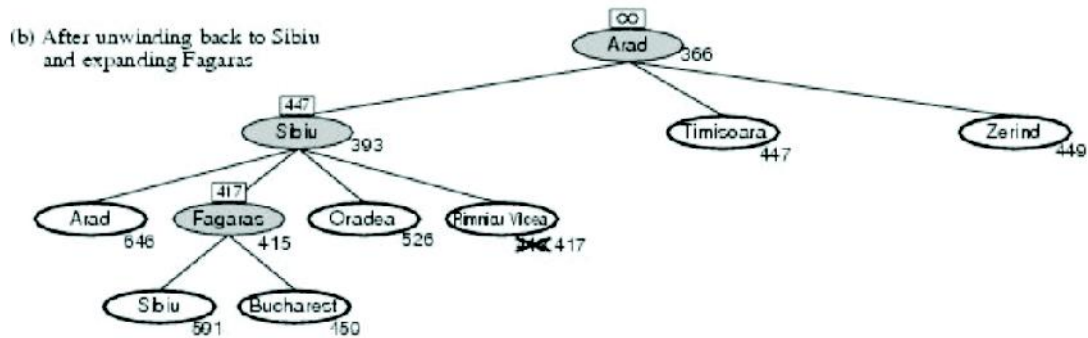
-Keeps track of the value of the best alternative path (**f -limit**) available from any ancestor of the current node.

-If current exceeds the stored limit, recursion unwinds back to the alternative path.

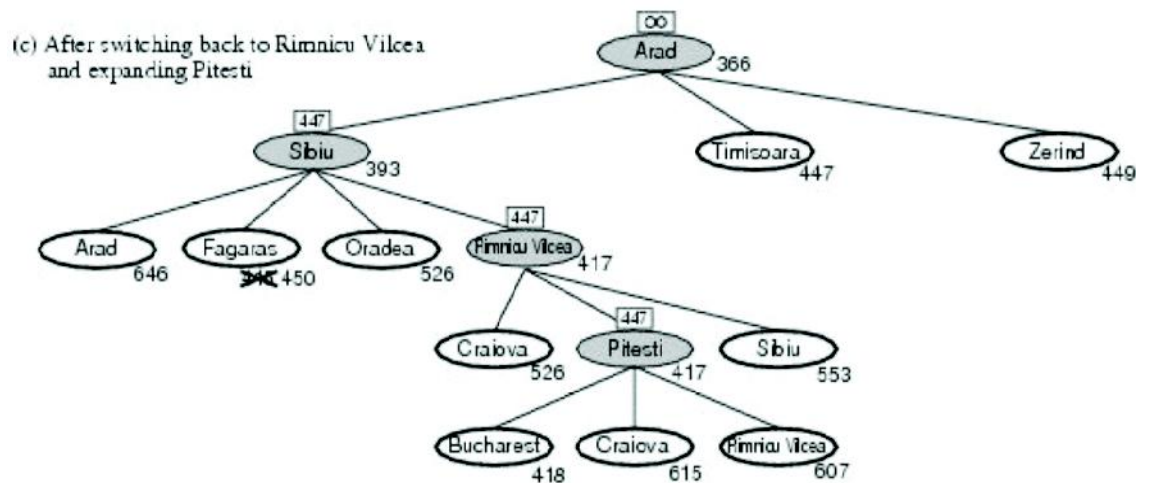
-Using the map above, moving from Arad to Bucharest.



- Path until Rumnicu Vilcea is already expanded
- Above node; f -limit for every recursive call is shown on top.
- Below node: $f(n)$ up to the given node.
- The path is followed until Pitesti



(1) Has a f -value worse than the f -limit. \rightarrow Backtrack



-Backtrack over Rimnicu Vilcea and store f -value for best child (Pitesti w/ value 417)

- *Best* is now Fagaras. *Best* child value is 450
But 450 is greater than 417 \rightarrow backtrack!
- Backtrack over Fagaras and store f -value for best child (bucharest=450)
- *Best* is now Rimnicu Viclea (again 447).
- Subtree is again expanded
- Best *alternative* subtree is now through Timisoara (447).
- Solution is found since because $447(\text{dist. at Timisoara}) > 417(\text{dist. at Pitetsi})$.

Characteristics

- Optimal if the heuristic function is admissible (optimistic)
- May explore same states several times, time consuming
- Saves space

Students to Look at:

- **Iterative Deepening (IDA*) Improved depth first search**
- **Simplified Memory Bounded (SMA*) Improved A***

Local Search Algorithms (Iterative improvement search)

-Only interested in solution, not path.

-Employed in optimization problems, e.g. Find the min(max) objective function.

Local search begins from an arbitrary state in the search space and looks for an improvement in the neighborhood of that state, until no improvement can be found.

(Iterative improvement)

Local search algorithms: keep a single "current" state, try to improve it according to an objective function.

Local Search Algorithms employ special heuristics.

- A heuristic is a simplification or educated guess that limits the search for solutions in domains that are difficult or poorly understood.
- It cannot be computed from problem definition itself
- A heuristic $h(n)$ is admissible if for every node n , $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n .
- An admissible heuristic never overestimates the cost to reach the goal, i.e., it is optimistic
- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)

Why Local Search?

- Uses little memory
- Finds reasonable solutions in large infinite spaces

Examples of Local Search Algorithms

1. Hill Climbing Search Algorithm (First Choice Hill Climbing)

function Hill-Climbing(*problem*) **returns** a state that is a local maximum

inputs: *problem*, a problem

local variables: *current*, a node

neighbor, a node

current \leftarrow Make-Node(Initial-State[*problem*])

loop do

neighbor \leftarrow a highest-valued successor of *current*

if Value[*current*] < Value[*neighbor*] then

current \leftarrow *neighbor*

else return STATE[*current*]

end

-This is a simplification of Best-First Search

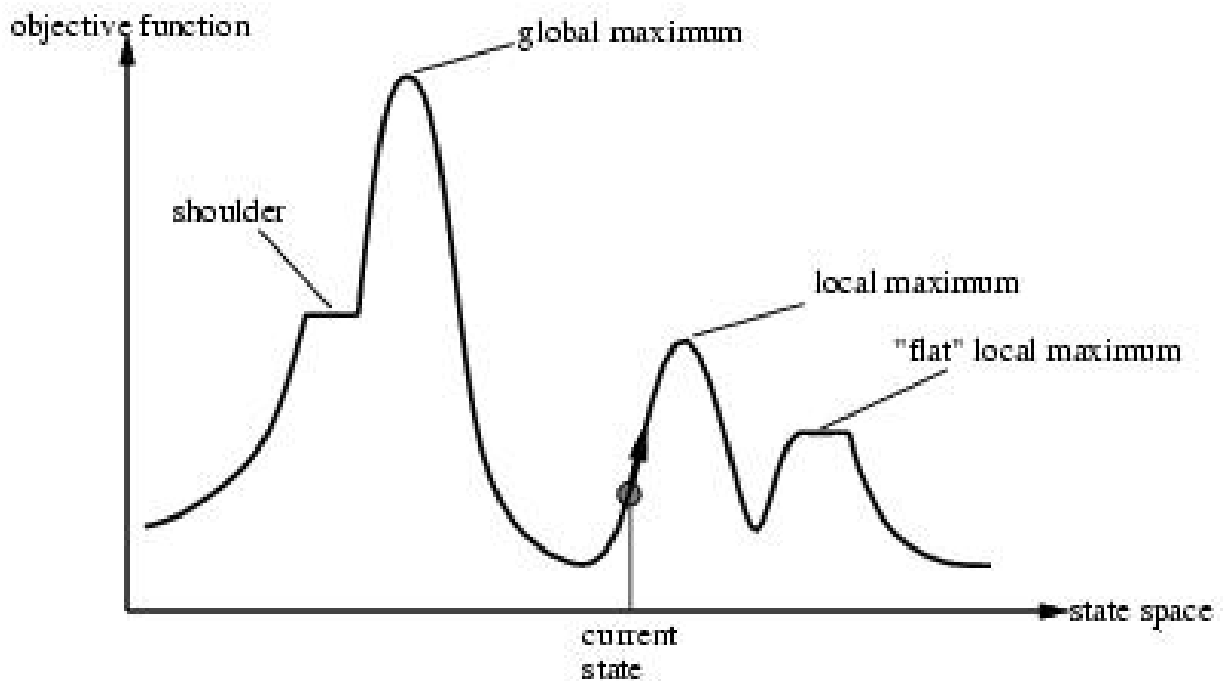
-Just keep track of the one state you are considering, and the path that got you there from the initial state.

-At every state choose the state that leads you closer to the goal (according to the heuristic estimate), and continue from there.

Hill climbing search works well (and is fast, and takes little memory) if an accurate heuristic measure is available in the domain, and if there are no local maxima.

In general the idea is to:

- Start from initial state, loop over operators, generate a state for the current operator. If the newly generated state is better than the current state, go to that state, repeat until a goal is found.
- **Dis-Advantages :** susceptible to local maxima, plateau's and ridges.
- Remember, from any given node, we go to the next node that is better than the current node. If there is no node better than the present node, then hill climbing halts, i.e. has found a solution.



Hill-climbing is a greedy strategy. The upside is it may make rapid progress towards the best state. The downside is that it can halt at a *local maximum*. A local maximum is a state that is better than all its successors but worse than the global maximum.

One workaround is to allow *sideway moves*, i.e. to allow the algorithm to choose a successor that has the same value as the current state. This can help a search get off a shoulder (why?), but the search becomes infinite on a plateau (why?). So a limit must be placed on the number of consecutive sideways moves.

The most sophisticated versions of this idea go under the name ***tabu search***. Tabu search is hill-climbing search, often with complete-state formulations, in which sideways moves are allowed. However, a fixed size memory is maintained of the most recently visited states. This is called the *tabu list*. The current state may not be replaced by any state that is currently on the tabu list.

Allowing sideway moves still doesn't solve the problem of other kinds of local maxima.

One possible solution is **random re-start hill-climbing**. In random re-start hill-climbing, a series of hill-climbing searches are executed from randomly-chosen initial states. In the limit (if enough random re-starts are done), this should find an optimal solution.

Another possible solution is to even allow *downwards moves*, i.e. if none of the successors is better than the current state, but the time limit is not yet reached, allow the algorithm to choose one of these successors that worsens the value. If you do this, you should store the current state before making the downwards move. Next time you reach a maximum, you can see whether the new maximum is better or worse than the one you stored earlier, and work with the better of the two. (This is not the same as having an agenda. Here we are remembering just one state: the best (local) maximum seen so far.)

Lecture 7

Knowledge & Reasoning

Perspectives of Knowledge Representation

a) KR as applied epistemology (study of the meaning of knowledge)

All intelligent activity presupposes knowledge. Knowledge is represented in a *knowledge base*, which consists of knowledge structures (typically symbolic) and programs

b) Knowledge Representation (KR) as a tell-ask module:

This is the lowest expectation out of a knowledge representation module in any AI system. Any KR system should provide at least two operations:

- TELL(K, f) Given a knowledge base K , the fact f is added to it resulting in a new knowledge base, say K' .
- ASK(K, f) The knowledge base K is being queried about a fact f . The answer, depending upon the KR paradigm (see below) used, may be *yes*, *no*, *unknown*, *yes with a confidence factor of A*, ...etc.

c) KR as the embodiment of AI systems:

The *connectionist view*. This approach takes the view that there are several identical interconnected units that are collectively responsible for representing various concepts. A concept is represented in a distributed sense (as opposed to local) and is indicated by an evolving pattern of activity over a collection of units.

Knowledge Representation and Reasoning

In **knowledge representation**, the main aim is to solve the problems of

- (1) how to represent the knowledge one has about a problem domain, and
- (2) how to reason using that knowledge in order to answer questions or make decisions

Knowledge representation deals with the problem of how to model the world sufficiently for intelligent action

Knowledge is represented in a *knowledge base*, which consists of knowledge structures (typically symbolic) and programs

Logic is one of the oldest representation languages studied for AI, and is the foundation for many existing systems that use logic as either inspiration or the basis for the tools in that system

- For a knowledge-based intelligent agent, we need:
 - To represent knowledge about the world in a *formal language*
 - To reason about the world using *inferences* in the language
 - To decide what action to take by *inferring* that the selected action is good

Knowledge Representation (KR) Paradigms

Key KR paradigms:

a) Procedural Knowledge: Knowledge is encoded in functions/procedures. e.g:

```
function Person(X) return boolean is  
if (X = ``Socrates") or (X = ``Hillary") then return true  
else return false;
```

```
function Mortal(X) return boolean is  
return person(X);
```

b) Semantic Networks: A compromise between declarative and procedural schemes. Knowledge is represented in a labeled, directed graph whose nodes represent concepts and entities, while its arcs represent relationships between these entities and concepts.

c) Frames: Much like a semantic network except each node represents prototypical concepts and/or situations. Each node has several property *slots* whose values may be specified or *inherited* by default.

d) Logic: A way of declaratively representing knowledge. e.g

```
person(Socrates).  
person(Hillary).  
forall X [person(X) ---> mortal(X)]
```

e) Decision Trees: Concepts are organized in the form of a tree.

f) Statistical Knowledge: The use of *certainty factors*, *Bayesian Networks*, *Dempster-Shafer Theory*, *Fuzzy Logics*, ..., etc.

g) Rules: The use of *Production Systems* to encode condition-action rules (as in expert systems).

h) Parallel Distributed processing: The use of connectionist models.

i) Sub-sumption Architectures: Behaviors are encoded (represented) using layers of simple (numeric) finite-state machine elements.

j) Hybrid Schemes: Any representation formalism employing a combination of KR schemes.

Representation Languages

- Fundamental problem of designing a knowledge representation language is the fundamental tradeoff between
- (1) a language that is **expressive** enough to represent the important objects and relations in a problem domain, yet
- (2) allows for a **tractable** (i.e., efficient) means of reasoning and answering questions about implicit information in a reasonable amount of time
-
- **Logic** is a formal system in which the formulas or sentences have true or false values. It is a well-studied, general-purpose language for describing what's true and false in the world, along with mechanical procedures that can operate on sentences in the language to perform reasoning (i.e., to determine what "implicitly follows" from what is explicitly represented)
- **A logic** includes:
 - **Syntax:** Specifies the symbols in the language and how they can be combined to form sentences. Hence facts about the world are represented as sentences in logic
 - **Semantics:** Specifies what facts in the world a sentence refers to. Hence, also specifies how you assign a truth value to a sentence based on its meaning in the world. A **fact** is a claim about the world, and may be true or false.
 - **Inference Procedure:** Mechanical method for computing (deriving) new (true) sentences from existing sentences

Properties of Logic

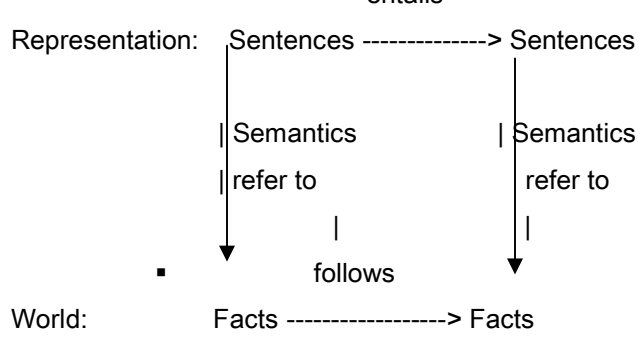
Declarative: i. e. pieces of syntax correspond to facts i.e. allows **partial/disjunctive/negated** information (unlike most data structures and databases)

Compositional: meaning of **A** \circ **B** is derived from meaning of **A** and of **B**

Meaning in propositional logic is **context-independent** (unlike natural language, where meaning depends on context)

Has very **limited expressive power** (unlike natural language) e.g., cannot say "pits cause breezes in adjacent squares" except by writing one sentence for each square.

- **Facts** in logic are claims about the world that are True or False, whereas a **representation** is an expression (sentence) in some language that can be encoded in a computer program and stands for the objects and relations in the world
- We need to ensure that the representation is consistent with reality, so that the following figure holds:
-

-
- Representation: 
 -
 -
 -
 -
 -
 - World: Facts -----> Facts
 -
 - **Truth:** A sentence is True if the state of affairs it describes is actually the case in the world. So, truth can only be assessed with respect to the semantics. Yet the computer does not know the semantics of the knowledge representation language, so we need some way of performing inferences to derive valid conclusions even when the computer does not know what the semantics (the interpretation) is
 -

To build a logic-based representation:

- User defines a set of primitive symbols and the associated semantics
- Logic defines the ways of putting these symbols together so that the user can define legal sentences in the language that represent true facts in the world
- Logic defines ways of inferring new sentences from existing ones

In (PL)

- A user defines a set of propositional **symbols**, like *P* and *Q*. User defines the semantics of each of these symbols. For example,
 - *P* means "It is hot"
 - *Q* means "It is humid"
 - *R* means "It is raining"
- A **sentence** (also called a formula or well-formed formula or wff) is defined as:
 1. A symbol
 2. If *S* is a sentence, then $\sim S$ is a sentence.
 3. If *S* and *T* are sentences, then $(S \vee T)$, $(S \wedge T)$, $(S \Rightarrow T)$, and $(S \Leftrightarrow T)$ are also sentences,
 4. A finite number of applications of (1)-(3)
- Examples of PL sentences:
 - $(P \wedge Q) \Rightarrow R$ (here meaning "If it is hot and humid, then it is raining")
 - $Q \Rightarrow P$ (here meaning "If it is humid, then it is hot")
 - *Q* (here meaning "It is humid.")

- Given the truth values, of all of the constituent symbols in a sentence, that sentence can be "evaluated" to determine its truth value (True or False). This is called an **interpretation** of the sentence.
-
- A **model** is an interpretation (i.e., an assignment of truth values to symbols) of a set of sentences such that each sentence is True. A model is just a formal mathematical structure that "stands in" for the world.
-
- A **valid** sentence (also called a **tautology**) is a sentence that is True under *all* interpretations. E.g. "It's raining or it's not raining."
-
- An **inconsistent** sentence (also called **un-satisfiable** or a **contradiction**) is a sentence that is False under *all* interpretations. Hence the world is never like what it describes. For example, "It's raining and it's not raining."
- Sentence P **entails** sentence Q, written $P \models Q$, means that whenever P is True, so is Q. In other words, all models of P are also models of Q

Logical (Deductive) Inference

Let $KB = \{ S_1, S_2, \dots, S_M \}$ be the set of all sentences in our Knowledge Base, where each S_i is a sentence in Propositional Logic. Let $\{ X_1, X_2, \dots, X_N \}$ be the set of all the symbols (i.e., variables) that are contained in all of the M sentences in KB. Say we want to know if a goal (aka query, conclusion, or theorem) sentence G follows from KB.

Since the computer doesn't know the **interpretation** of these sentences in the world, we don't know whether the constituent symbols represent facts in the world that are True or False. So, instead, consider *all* possible combinations of truth values for all the symbols, hence enumerating all logically distinct cases:

X_1	X_2	..	X_N	S_1	S_2	...	S_M	$S_1 \wedge S_2 \wedge \dots \wedge S_M$	G	$(S_1 \wedge \dots \wedge S_M) \Rightarrow G$
F	F	...	F							
F	F	...	T							
T	T	..	T							

There are 2^N rows in the table.

- Each row corresponds to an equivalence class of worlds that, under a given interpretation, have the truth values for the N symbols assigned in that row.
- The **models** of KB are the rows where the third-to-last column is *true*, i.e., where all of the sentences in KB are *true*.
- A sentence R is **valid** if and only if it is true under all possible interpretations, i.e., if the entire column associated with R contains all *true* values.
- Since we don't know the semantics and therefore whether each symbol is True or False, to determine if a sentence G is **entailed** by KB, we must determine if **all** models of KB are also models of G . That is, whenever KB is true, G is true too. In other words, whenever the third-from-last column has a T, the same row in the

second-from-last column also has a T. But this is logically equivalent to saying that the sentence $(KB \Rightarrow G)$ is valid (by definition of the "implies" connective).

-
- In other words, if the last column of the table above contains only *True*,
- then **KB entails G**; or conclusion G logically follows from the premises in KB, no matter what the interpretations (i.e., semantics) associated with all of the sentences!
- The truth table method of inference is **complete** for Propositional Logic because we can always enumerate all 2^n rows for the n propositional symbols that occur. But this is exponential in n . In general, it has been shown that the problem of checking if a set of sentences in PL is satisfiable is NP-complete. (The truth table method of inference is *not* complete for FOL (First-Order Logic).)

Recap

Truth Table of the Binary Logical Connectives

P	Q	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
T	T	T	T	T	T
T	F	F	T	F	F
F	T	F	T	T	F
F	F	F	F	T	T

Example

Using the "weather" sentences from above, let $KB = (((P \wedge Q) \Rightarrow R) \wedge (Q \Rightarrow P) \wedge Q)$ corresponding to the three facts we know about the weather:

- (1) "If it is hot and humid, then it is raining,"
- (2) "If it is humid, then it is hot," and
- (3) "It is humid."

Now let's ask the query "Is it raining?" That is, is the query sentence R entailed by KB? Using the truth-table approach to answering this query we have:

P	Q	R	$(P \wedge Q) \Rightarrow R$	$Q \Rightarrow P$	Q	KB	R	$KB \Rightarrow R$
T	T	T	T	T	T	T	T	T
T	T	F	F	T	T	F	F	T
T	F	T	T	T	F	F	T	T
T	F	F	T	T	F	F	F	T
F	T	T	T	F	T	F	T	T
F	T	F	T	F	T	F	F	T
F	F	T	T	T	F	F	T	T

F F F T T F F F T

Hence, in this problem there is only one model of KB, when P, Q, and R are all True. And in this case R is also True, so R is entailed by KB. Also, you can see that the last column is all True values, so the sentence $KB \Rightarrow R$ is valid.

Instead of creating a truth table, a **proof procedure** or **inference procedure** that uses **sound rules of inference** to deduce (i.e., derive) new sentences that are true in all cases where the premises are true, can be used. For example, consider the following:

P	Q	$P \Rightarrow Q$	$P \wedge (P \Rightarrow Q)$	Q	$(P \wedge (P \Rightarrow Q)) \Rightarrow Q$
F	F	F	T	F	T
F	T	F	T	F	T
T	F	T	F	F	T
T	T	T	T	T	T

Since whenever P and $P \Rightarrow Q$ are both true (last row only), Q is true too, Q is said to be **derived** from these two premise sentences. We write this as $KB \vdash Q$. This local pattern referencing only two of the M sentences in KB is called the **Modus Ponens** inference rule. The truth table shows that this inference rule is **sound**. It specifies how to make one kind of step in deriving a conclusion sentence from a KB.

Therefore, given the sentences in KB, construct a **proof** that a given conclusion sentence can be derived from KB by applying a sequence of sound inferences using either sentences in KB or sentences derived earlier in the proof, until the conclusion sentence is derived.

Sound Rules of Inference

Here are some examples of sound rules of inference. Each can be shown to be sound once and for all using a truth table. The left column contains the premise sentence(s), and the right column contains the derived sentence. We write each of these derivations as $A \vdash B$, where A is the premise and B is the derived sentence.

Name	Premise(s)	Derived Sentence
Modus Ponens	$A, A \Rightarrow B$	B
And Introduction	A, B	$A \wedge B$
And Elimination	$A \wedge B$	A
Double Negation	$\sim\sim A$	A
Unit Resolution	$A \vee B, \sim B$	A

Resolution

$A \vee B, \sim B \vee C$

$A \vee C$

Using Inference Rules to Prove a Query/Goal/Theorem

A proof is a sequence of sentences, where each sentence is either a premise or a sentence derived from earlier sentences in the proof by one of the rules of inference. The last sentence is the query (also called goal or theorem) that we want to prove.

Example for the "weather problem" given above.

- | | |
|---------------------------------|------------------------|
| 1. Q | Premise |
| 2. $Q \Rightarrow P$ | Premise |
| 3. P | Modus Ponens (1,2) |
| 4. $(P \wedge Q) \Rightarrow R$ | Premise |
| 5. $P \wedge Q$ | And Introduction (1,3) |
| 6. R | Modus Ponens (4,5) |

Two Important Properties for Inference

- **Soundness:** If $KB \vdash Q$ then $KB \models Q$
That is, if Q is derived from a set of sentences KB using a given set of rules of inference, then Q is entailed by KB. Hence, inference produces only real entailments, or any sentence that follows deductively from the premises is valid.
- **Completeness:** If $KB \models Q$ then $KB \vdash Q$
That is, if Q is entailed by a set of sentences KB, then Q can be derived from KB using the rules of inference. Hence, inference produces *all* entailments, or all valid sentences can be proved from the premises.

Propositional Logic is Too Weak a Representational Language

(PL) is not a very expressive language because: -

Hard to identify "individuals." e.g., Mary, 3

- Can't directly talk about properties of individuals or relations between individuals.
e.g., tall(Bill)
- Generalizations, patterns, regularities can't easily be represented. E.g., all triangles have 3 sides

Consider the problem of representing the following information:

Every person is mortal.

Confucius is a person.

Confucius is mortal.

How can these sentences be represented so that we can infer the third sentence from the first two? In PL we have to create propositional symbols to stand for all or part of each sentence. For example, we might do:

Person \Rightarrow Mortal

Person-Confucius

Mortal-Confucius

That is, we have used four symbols to represent the three given sentences. But, given this representation, the third sentence is *not* entailed by the first two.

A different representation would be to use three symbols to represent the three sentences as:

Person \Rightarrow Mortal

Confucius \Rightarrow Person

Confucius \Rightarrow Mortal

In this case the third sentence *is* entailed by the first two, but we needed an explicit symbol, Confucius, to represent an individual who is a member of the classes "person" and "mortal." So, to represent other individuals we must introduce separate symbols for each one, with means for representing the fact that all individuals who are "people" are also "mortal." **First-Order Logic** (abbreviated **FOL** or **FOPC**) is expressive enough to concisely represent this kind of situation. **Read about the Wampus World**

Lecture 8

First-Order Logic (FOL) Syntax

- User defines some primitives:
 - **Constant symbols** (i.e., the "individuals" in the world) e.g., Mary, 3
 - **Function symbols** (mapping individuals to individuals)
 - e.g., father-of(Mary) = John, color-of(Sky) = Blue
 - **Predicate symbols** (mapping from individuals to truth values) e.g., greater(5,3), green(Grass), color(Grass, Green)

FOL primitives:

Variable symbols. e.g., x, y

Connectives. Same as in PL: not (\sim), and (\wedge), or (\vee), implies (\Rightarrow), if and only if (\Leftrightarrow)

Quantifiers: Universal (\forall) and Existential (\exists)

Universal quantification corresponds to conjunction ("and") in that $(\forall x)P(x)$ means that P holds for all values of x in the domain associated with that variable.

e.g. $(\exists x) \text{dolphin}(x) \Rightarrow \text{mammal}(x)$

Existential quantification corresponds to disjunction ("or") in that $(\exists x)P(x)$ means that P holds for some value of x in the domain associated with that variable.

e.g., $(\exists x) \text{mammal}(x) \wedge \text{lays-eggs}(x)$

Universal quantifier normally used with "implies" to form "if-then rules."

e.g., $(\forall x) \text{MIM703student}(x) \Rightarrow \text{smart}(x)$ means "All MIM703 students are smart."

Existential quantifier normally used with "and" to specify a list of properties or facts about an individual.

e.g., $(\exists x) \text{MIM703student}(x) \wedge \text{smart}(x)$ means

" There is an MIM703 student who is smart."

Switching the order of universal quantifiers does not change the meaning: $(\forall x)(\forall y)P(x,y)$ is logically equivalent to $(\forall y)(\forall x)P(x,y)$. Similarly, you can switch the order of existential quantifiers.

Switching the order of universals and existentials, *does* change meaning:

Everyone likes someone: $(\forall x)(\exists y)\text{likes}(x,y)$

Someone is liked by everyone: $(\exists y)(\forall x)\text{likes}(x,y)$

Building sentences in FOL

Sentences are built up from terms and atoms:

A **term** (denoting a real-world individual) is a constant symbol, a variable symbol, or an n-place function of n terms. For example, x and $f(x_1, \dots, x_n)$ are terms, where each x_i is a term.

An **atom** (which has value true or false) is either an n-place predicate of n terms, or, if P and Q are atoms, then $\sim P$, $P \vee Q$, $P \wedge Q$, $P \Rightarrow Q$, $P \Leftrightarrow Q$ are atoms

A **sentence** is an atom, or, if P is a sentence and x is a variable, then $(\forall x)P$ and $(\exists x)P$ are sentences

A **well-formed formula (wff)** is a sentence containing no "free" variables. i.e., all variables are "bound" by universal or existential quantifiers.

e.g., $(\forall x)P(x,y)$ has x bound as a universally quantified variable, but y is free.

Translating English to FOL

Every gardener likes the sun.

$(\forall x) \text{gardener}(x) \Rightarrow \text{likes}(x, \text{Sun})$

You can fool all of the people some of the time.

$(\forall x) (\text{person}(x) \Rightarrow (\exists t) (\text{time}(t) \wedge \text{can-fool}(x,t)))$

All purple mushrooms are poisonous.

$(\forall x) (\text{mushroom}(x) \wedge \text{purple}(x)) \Rightarrow \text{poisonous}(x)$

No purple mushroom is poisonous.

$\sim(\exists x) \text{purple}(x) \wedge \text{mushroom}(x) \wedge \text{poisonous}(x)$

or, equivalently,

$(\forall x) (\text{mushroom}(x) \wedge \text{purple}(x)) \Rightarrow \sim\text{poisonous}(x)$

Deb is not tall.

$\sim\text{tall}(\text{Deb})$

X is above Y if X is directly on top of Y or else there is a pile of one or more other objects directly on top of one another starting with X and ending with Y .

$(\forall x)(\forall y) \text{above}(x,y) \Leftrightarrow (\text{on}(x,y) \vee (\exists z) (\text{on}(x,z) \wedge \text{above}(z,y)))$

Inference Rules for FOL

Inference rules for PL apply to FOL as well. e.g., Modus Ponens, And-Introduction, And-Elimination, etc.

New (sound) inference rules for use with quantifiers:

Universal Elimination

If $(\forall x)P(x)$ is true, then $P(c)$ is true, where c is a constant in the domain of x .

e.g. from $(\forall x)\text{eats}(\text{Ziggy}, x)$ we can infer $\text{eats}(\text{Ziggy}, \text{IceCream})$.

The variable symbol can be replaced by any ground term, i.e., any constant symbol or function symbol applied to ground terms only.

Existential Introduction

If $P(c)$ is true, then $(\exists x)P(x)$ is inferred.

e.g., from $\text{eats}(\text{Ziggy}, \text{IceCream})$ we can infer $(\exists x)\text{eats}(\text{Ziggy}, x)$. All instances of the given constant symbol are replaced by the new variable symbol. Note that the variable symbol cannot already exist anywhere in the expression.

Existential Elimination

From $(\exists x)P(x)$ infer $P(c)$. For example, from $(\exists x)\text{eats}(\text{Ziggy}, x)$ infer $\text{eats}(\text{Ziggy}, \text{cheese})$.

Note that the variable is replaced by a *brand new* constant that does not occur in this or *any other* sentence in the Knowledge Base.

Generalized Modus Ponens (GMP)

Combines And-Introduction, Universal-Elimination, and Modus Ponens

Example: from $P(c)$, $Q(c)$, and $(\forall x)(P(x) \wedge Q(x)) \Rightarrow R(x)$, derive $R(c)$

In general, given atomic sentences P_1, P_2, \dots, P_N , and implication sentence

$(Q_1 \wedge Q_2 \wedge \dots \wedge Q_N) \Rightarrow R$, where Q_1, \dots, Q_N and R are atomic sentences, and

$\text{subst}(\text{Theta}, P_i) = \text{subst}(\text{Theta}, Q_i)$ for $i=1, \dots, N$, derive new sentence: $\text{subst}(\text{Theta}, R)$

$\text{subst}(\text{Theta}, \alpha)$ denotes the result of applying a set of substitutions defined by Theta to the sentence α

A substitution list $\text{Theta} = \{v_1/t_1, v_2/t_2, \dots, v_n/t_n\}$ means to replace all occurrences of variable symbol v_i by term t_i . Substitutions are made in left-to-right order in the list.

e.g: $\text{subst}(\{x/\text{IceCream}, y/\text{Ziggy}\}, \text{eats}(y,x)) = \text{eats}(\text{Ziggy}, \text{IceCream})$

Automated Inference for FOL

Generalized Modus Ponens is not complete for FOL but is complete for KBs containing only **Horn clauses**

A Horn clause is a sentence of the form:

$(\forall x) (P_1(x) \wedge P_2(x) \wedge \dots \wedge P_n(x)) \Rightarrow Q(x)$

where there are 0 or more P_i 's, and the P_i 's and Q are positive (i.e., un-negated) literals

Proofs start with the given axioms/premises in KB, deriving new sentences using GMP until the goal/query sentence is derived.

This defines a **forward chaining** inference procedure because it moves "forward" from the KB to the goal.

e.g.: KB = 1. All cats like fish, 2. cats eat everything they like, and 3. Ziggy is a cat.

In FOL, KB = $(\forall x) \text{cat}(x) \Rightarrow \text{likes}(x, \text{Fish})$
 $(\forall x)(\forall y) (\text{cat}(x) \wedge \text{likes}(x,y)) \Rightarrow \text{eats}(x,y)$ cat(Ziggy)

Goal query: Does Ziggy eat fish?

Proof:

Use GMP with (1) and (3) to derive: 4. likes(Ziggy, Fish)

Use GMP with (3), (4) and (2) to derive eats(Ziggy, Fish)

So, Yes, Ziggy eats fish.

Backward-chaining deduction using GMP is complete for KBs containing only Horn clauses. Proofs start with the goal query, find implications that would allow you to prove it, and then prove each of the antecedents in the implication, continuing to work "backwards" until we get to the axioms, which we know are true.

e.g: Does Ziggy eat fish?

To prove eats(Ziggy, Fish), first see if this is known from one of the axioms directly.

Here it is not known, so see if there is a Horn clause that has the consequent (i.e., right-hand side) of the implication matching the goal. Here,

Proof:

Goal matches RHS of Horn clause (2), so try and prove new sub-goals cat(Ziggy) and likes(Ziggy, Fish) that correspond to the LHS of (2) cat(Ziggy) matches axiom (3), so we've "solved" that sub-goal likes(Ziggy, Fish) matches the RHS of (1), so try and prove cat(Ziggy) cat(Ziggy) matches (as it did earlier) axiom (3), so we've solved this sub-goal

There are no unsolved sub-goals, so we're done. Yes, Ziggy eats fish.

Resolution Refutation Procedure (aka Resolution Procedure)

Resolution procedure is a sound and complete inference procedure for FOL

Resolution procedure uses a single rule of inference: the Resolution Rule (RR), which is a generalization of the same rule used in PL.

Resolution Rule for Propositional Logic:

From sentence $P_1 \vee P_2 \vee \dots \vee P_n$ and sentence $\sim P_1 \vee Q_2 \vee \dots \vee Q_m$ derive **resolvent**

sentence: $P_2 \vee \dots \vee P_n \vee Q_2 \vee \dots \vee Q_m$

Examples

From P and $\sim P \vee Q$, derive Q (Modus Ponens)

From $(\sim P \vee Q)$ and $(\sim Q \vee R)$, derive $\sim P \vee R$

From P and $\sim P$, derive False

From $(P \vee Q)$ and $(\sim P \vee \sim Q)$, derive True

Resolution Rule for FOL:

Given sentence $P_1 \vee \dots \vee P_n$ and sentence $Q_1 \vee \dots \vee Q_m$ where each P_i and Q_i is a **literal**, i.e., a positive or negated predicate symbol with its terms, if P_j and $\sim Q_k$ **unify** with

substitution list Theta, then derive the resolvent sentence:

$\text{subst}(\text{Theta}, P_1 \vee \dots \vee P_{j-1} \vee P_{j+1} \vee \dots \vee P_n \vee Q_1 \vee \dots \vee Q_{k-1} \vee Q_{k+1} \vee \dots \vee Q_m)$

Example

From clause $P(x, f(a)) \vee P(x, f(y)) \vee Q(y)$ and clause $\sim P(z, f(a)) \vee \sim Q(z)$, derive resolvent clause $P(z, f(y)) \vee Q(y) \vee \sim Q(z)$ using $\text{Theta} = \{x/z\}$

Unification

Unification is a "pattern matching" procedure that takes two atomic sentences, called **literals**, as input, and returns "failure" if they do not match and a substitution list, Theta, if they do match. That is, $\text{unify}(p, q) = \text{Theta}$ means $\text{subst}(\text{Theta}, p) = \text{subst}(\text{Theta}, q)$ for two atomic sentences p and q.

Theta is called the **most general unifier (mgu)**

All variables in the given two literals are implicitly universally quantified

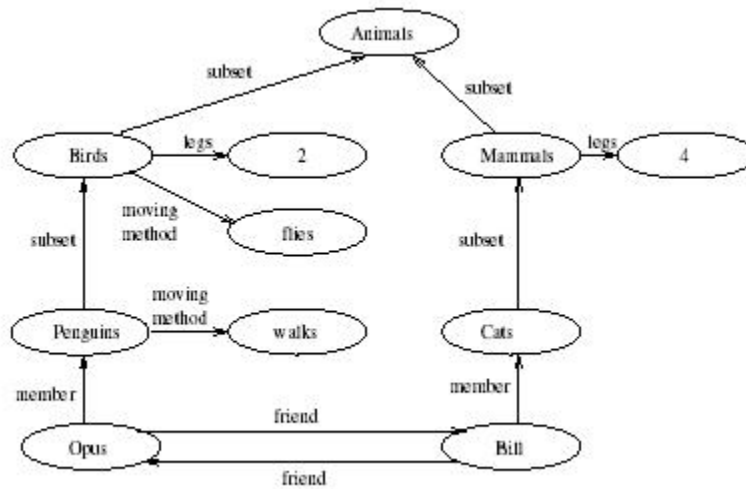
To make literals match, replace (universally-quantified) variables by terms

Examples

Literal 1	Literal 2	Result of Unify
Parents(x, father(x), mother(Bill))	parents(Bill, father(Bill), y)	{ x/Bill, y/mother(Bill) }
Parents(x, father(x), mother(Bill))	parents(Bill, father(y), z)	{ x/Bill, y/Bill, z/mother(Bill) }
Parents(x, father(x), mother(Jane))	parents(Bill, father(y), mother(y))	Failure

3. Semantic Networks

- Semantic net is a *labelled graph*.
- Models the associations between ideas that people maintain.
- **nodes** in the graph represent *objects, concepts, or situations*;
- **arcs** in graph represent *relationships between objects*.



Key types of arc (relationships)

x is a subset of y

x is a kind of y (ako relationship) e.g. penguin ako bird)

x is a member of y

x is a y (is-a relationship) e.g. opus is-a penguin

x is R-related to y

e.g. bill friend opus (friend is the relationship)

Inheritance

Inheritance is one of the main kinds of reasoning done in semantic nets

The subset (ako) relation is often used to link a class and its super-class.

Some links (e.g. legs) are inherited along subset paths

The semantics of a semantic net can be relatively informal or very formal

Often defined at the implementation level

A node can have any number of super-classes that contain it, enabling a node to inherit properties from multiple parent nodes and their ancestors in the network. This can cause conflicting inheritance.

Advantages of Semantic Nets

- Easy to visualize
- Relationships can be arbitrarily defined by the knowledge engineer
- Formal definitions of semantic networks have been developed.

- Related knowledge is easily clustered.
- Efficient in space requirements
- Objects represented only once

Disadvantages of Semantic Nets

- Inheritance (particularly from multiple sources and when exceptions in inheritance are wanted) can cause problems.
- Facts placed inappropriately cause problems.
- No standards about node and arc values

Attempted Improvements (Net Variations)

- Building of search heuristics into the network.
- More sophisticated logical structure, involving partitioning.
- However these improvements meant that the formalism's original simplicity was lost.

Frames

Frames aka semantic net with *properties* and *methods*

Devised by Marvin Minsky, 1974.

Incorporates certain valuable human thinking characteristics:

Expectations, assumptions, stereotypes. Exceptions. Fuzzy boundaries between classes.

The essence of this form of knowledge representation is *typicality*, with exceptions, rather than *definition*.

Hierarchical structure

How Frames are Organised I

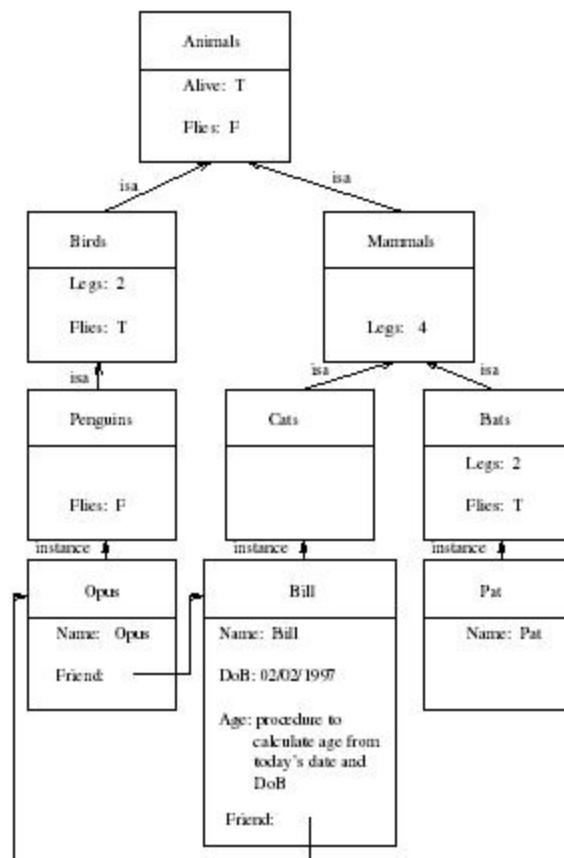
A frame can represent a specific entry, or a general concept

Each frame has:

A name

Slots (attributes) which have *values*

- a specific value
- a default value
- an inherited value
- a pointer to another frame
- a *procedure* that gives the value



Reasoning with Frames

Easy to answer questions such as *is x a y?*

Simply follow the instance and/or is-a links.

e.g. Is Opus a bird?

Also useful for *default* reasoning.

Simply *inherit* all default values that are not explicitly provided.

e.g How many legs does Opus have?

Frame Organization

In the higher levels of the frame hierarchy, typical knowledge about the class is stored.

In the lower levels, the value in a slot may be a specific value, to overwrite the value which would otherwise be inherited from a higher frame.

An instance of an object is joined to its class by an instance relationship.

A class is joined to its super class by an is-a relationship.

Frame Advantages

- Fairly intuitive for many applications
- Similar to human knowledge organization
- Suitable for causal knowledge
- Easy to include default information and detect missing values

- Easier to understand than logic or rules
- Very flexible

Frame Disadvantages

- No standards (slot-filler values)
- More of a general methodology than a specific representation:
- Frame for a class-room will be different for a professor and for a maintenance worker
- No associated reasoning/inference mechanisms

Common Problems with Frames & Semantic Nets

- Both frames and semantic nets are essentially *arbitrary*.
- Both are useful for representing certain sorts of knowledge.
- But both are essentially *ad hoc* & lack precise meaning, or *semantics*.
- Inference procedures poorly defined & justified.

Certainty Factors in Expert Systems

A number of errors contribute to uncertainty in the data from an expert system. The following error types are evident

- i) Ambiguity – something interpreted in a number of ways
- ii) Incomplete data- missing information
- iii) Incorrect – Wrong data, may be due to :
 - Human error, accidental misreading of data
 - Equipment malfunction, leading to wrong output, unreliable data, lack of output.
 - False Negative, rejection of a true hypothesis
 - False positive –acceptance of a wrong hypothesis
- iv) Measurement, with regards to precision & accuracy
- v) Random –due to the random nature of the system
- vi) Systematic –introduced due to some bias
- vii) Reasoning – inductive or deductive error

A number of approaches can be employed to deal with errors in the system, examples are:-

1. Certainty Factors

Certainty factors are values used to approximate the degree to which we think a rule in a rule-based system is correct. These values range from -1 and +1. The negative value indicates predominance of opposing evidence, while the positive value indicates a predominance of confirming evidence for the rule being correct.

e.g. If interest rates = fall (CF = 0.6) AND

Taxes = Reduced (CF = 0.8)

Then Stock Market = Rise (CF = 0.9)

The certainty factor of the stock market rising then becomes:

$$\text{Min}(0.6, 0.8) * 0.9 = 0.54.$$

If there are two rules talking about the same point then the certainty factor of the point is considered as the maximum of the CFs of the two rules.

The general approach for certainty factors in rules is:

1. From a rule, take the minimum CF of clauses connected by AND.
2. For OR connections in the rule, take the maximum CF value of all the AND clauses that are connected by ORs.
3. Multiply the final CF of the clauses by the CF of the rule.
4. If there is more than one rule leading to the same conclusion, take as the final CF the maximum CF values of all those rules.

Let us look at another example:

If A (CF = 0.3) AND (B (CF = 0.6)
THEN C (CF = 0.5)

If D (CF = 0.4) AND E (CF = 0.7)
THEN C (CF = 0.9)

The CF of C is the higher of the CFs of the two rules, as follows:

$$\begin{aligned} &\text{Max}((\text{min}(0.3, 0.6) * 0.5), \text{min}(0.4, 0.7) * 0.9)) \\ &= \text{Max}(0.15, 0.36) \\ &= 0.36 \end{aligned}$$

Example 2

If A (CF = 0.3) AND
B (CF = 0.6) OR
D (CF = 0.5)

$$\begin{aligned} &\text{Then C (CF = 0.4)} \\ &\text{Maximum}(\text{min}(0.3, 0.6), 0.5) * 0.4 \\ &= \text{max}(0.3, 0.5) * 0.4 \\ &= 0.5 * 0.4 = 0.2 \end{aligned}$$

Probability in Expert Systems

Probability (P) – study of degrees of randomness

$$= \frac{\text{Total number of ways a specific event can occur}}{\text{Total number of ways any event can occur}}$$

Probability and Bayes' Theorem

Classical probability theory (and in particular, Bayesian statistics) provides us with a statistical theory of evidence, which allows us to update the probabilities attached to

certain facts in the light of new evidence. The fundamental notion is that of conditional probability:

$$P(H | E)$$

This is read as the probability that a hypothesis H is true given that we have observed evidence E for the hypothesis. So, for example, we might be interested in the probability that a patient has measles given the knowledge that they have spots:

$$P(\text{patient-has-measles} | \text{patient-has-spots})$$

Sometimes we will know how likely some "evidence" is, if some hypothesis is true, but not the other way around. For example, we may know that 50% of people with measles have spots. We may also know that:

The only diseases that cause spots are measles, chickenpox and lassa fever.

60% of people with chickenpox have spots.

80% of people with lassa fever have spots.

There is a 1% chance of someone in a given population having measles (given no evidence for or against).

There is a 1% chance of them having chickenpox.

There is a 0.05% chance of them having lassa fever.

This can be represented more formally as:

$$P(\text{spots} | \text{measles}) = 0.5$$

$$P(\text{spots} | \text{chickenpox}) = 0.6$$

$$P(\text{spots} | \text{lassa}) = 0.8$$

$$P(\text{measles}) = 0.01$$

$$P(\text{chickenpox}) = 0.01$$

$$P(\text{lassa}) = 0.0005$$

From this we can calculate the probability that they have measles if they have spots (if we have no other evidence). Abbreviating things somewhat, for the above example we have:

Or using the actual probabilities:

$$\begin{aligned} P(m | s) &= 0.5 \times 0.01 / (0.5 \times 0.01 + 0.6 \times 0.01 + 0.8 \times 0.0005) \\ &= 0.44 \end{aligned}$$

Bayes' theorem is only valid if we know all the conditional probabilities relating to the evidence in question. In fact, as we consider more and more evidence it quickly becomes computationally intractable to use Bayes' theorem, quite apart from the problem of obtaining and representing all the conditional probabilities. Because of this, Bayes' theorem is rarely used. However, it is important as it is a well-known, sound way of dealing with the probabilities of hypotheses given evidence, and as such provides a kind of standard for assessing other approaches.