100-366-WK 15 APR 2012

09 Monday

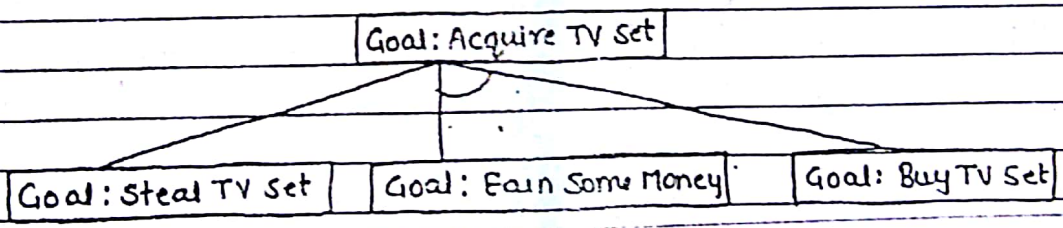
Problem Reduction

AND-OR Graphs.

When a problem can be divided into a set of sub problems, where each sub problem can be solved seprately. Another Kind of structure, the AND-OR graph (tree) is useful for representing the solution of problems that can be solved by decomposing them into set of smaller problems, all of which must then be solved.

This decomposition, or reduction generates arcs that we call AND arcs. One AND arc may point to any number of successor nodes, all of which must be solved. Just as in OR graph, several arcs may emerge from a single node, indicating a variety of ways in which the original problem might be solved. This is why the structure is called not simply an AND graph but rather an AND-OR graph.

Example of AND-OR Graph:



A Simple AND-OR Graph

In order to find solutions in an AND-OR Graph, we need an algorithm similar to best-first search but with the ability to handle the AND arcs appropriately.

= 1 + (Cost fun^n from S to C) + Cost fun^n from C to H + Cost function from H to T + Cost function from I to K)

= 1 + 6 + 5 + 7 + 1 = 20

Algorithm for A* algorithm method

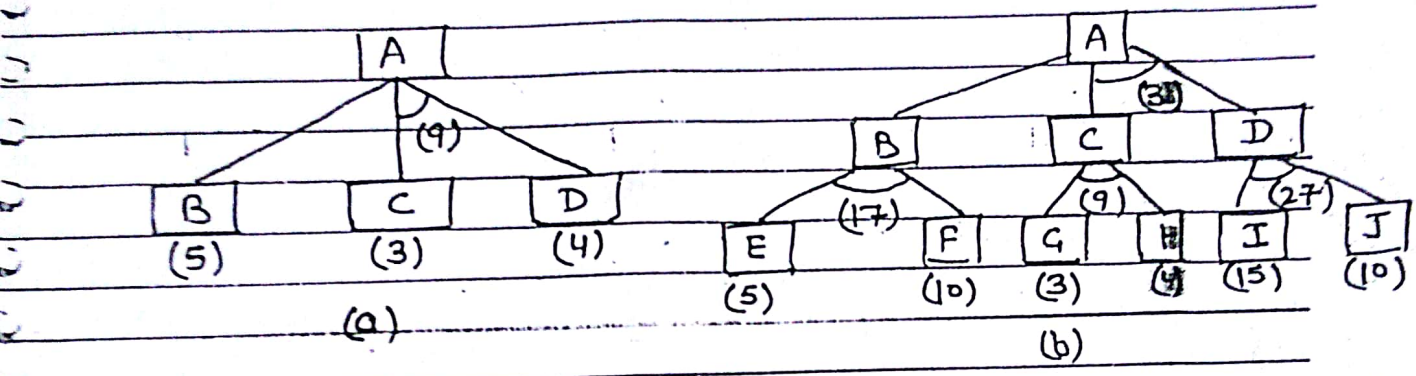
Step 1: Put the initial node on a list START.
Step 2: if (START) is empty) or (START = GOAL) terminate search.
Step 3: Remove the first node from START. Call this node a.
Step 4: if (a = GOAL) terminate search with sucess.
Step 5: Else if node a has successors, generate all of them. Estimate the fitness no. of the successors by totaling the evaluation fun^n and the cost-function value. Sort the list by fitness no.
Step 6: Name the new list as LIST 1 / START 1
Step 7: Replace START with START 1
Step 8: Goto Step 2

This algo is admissible. By admissible, one means that the algo is sure to find the most optimal path if one exists. This is possible only when the evaluation fun^n value NEVER OVERESTIMATES the distance of the node to the goal.

Sunday 08

Why Best-First Search algorithm is not adequate for searching AND-OR graph? or Example of understanding working of AND-OR Graph.

Consider below graph. The top node A, has been expanded, producing two-arcs, one leading to B and one leading to C and D. The numbers at each node represent the value of f' at that node.



We assume that every operation has a uniform cost, so each arc with a single successor has a cost of 1 and each AND arc with multiple successors has a cost of 1 for each of its components. If we use best first search, we must select the node c having the minimum h' value. But using the information now available, it would be better to explore the path going through B since to use C we must also use D, for a total cost of 9 (C+D+2) compared to the cost of 6 that we get by going through B.

The problem is that the choice of which node to expand next must depend not only on the f' value of that node but also on whether that node is part of the current best path from the initial node.

Consider another figure (b), The most promising node is G with an f' value of 3. It is even part of the most promising arc G-H, with a total cost of 9. But that arc is not part of the current path since to use

it with the arc I-J, with a cost of 27. The path from A, through B, to E and F is better, with a total cost of 18. So, we should not expand G next; rather we should examine either E or F.

The AO* Algorithm

The AO* algorithm will use a single structure GRAPH, representing the part of the search graph. Each node in the graph will point both down to its immediate successors and up to the immediate predecessors. Each node in the graph will also have associated with it an h' value.

Algorithm: AO*

1. Let GRAPH consist only of the node representing the initial node. (Call this node S.) Compute h'(s).

2. Until S is labeled SOLVED or until S's h' value becomes greater than FUTILITY limit. Repeat the following procedure

(a) Trace the labeled arcs from S and select the unexpanded node for expansion. Call the selected node NODE.

(b) Generate the successors of NODE. If there are none, then assign FUTILITY as the h' value of NODE. This is equivalent to saying that NODE is not solvable. If there are successors, then for each one that is not also an ancestor of NODE do the following:

(i) Add SUCCESSOR to GRAPH.

(ii) if SUCCESSOR is a terminal node, label it SOLVED and assign it an h' value of 0.

(iii) if SUCCESSOR is not a terminal node, compute its h' value.

Thursday 12 APR 2012

(c) Call cost revise(n) to propagate the newly information up the graph.

(d) Loop: Goto Step 2.

Algorithm: AO* Algorithm: Cost Revision(n).

1. Let S be a set of nodes that have been labeled SOLVED or whose h' values has been changed and so need to have values propagated back to their parents.

2. Select a node from set S ~~whose~~ none of descendants in GRAPH occur in S. and call it CURRENT node.

3. if CURRENT is an AND node with children $r_1, r_2, \ldots, r_k$.
Set $h'(\text{CURRENT}) = E[f(r_i) + C(\text{CURRENT}, r_i)]$

where $f(r_i)$ = cost of child node, and
$C(\text{CURRENT}, r_i)$ = cost of edge.
Mark the each edge to each child of CURRENT with a double line indicating it is SOLVED. if each child is labeled SOLVED, then label CURRENT as SOLVED.

4. if CURRENT is an OR node with children $r_1, r_2, \ldots, r_k$.
Set $h'(\text{current}) = \min\{f(r_i) + C(\text{CURRENT}, r_i)\}$
Mark the edge to the best child of CURRENT with a double line indicating it is SOLVED. if the marked child is labeled SOLVED, then label CURRENT as SOLVED.

5. if CURRENT has been solved or if the cost of CURRENT was just changed, then its new status must be propagated back up the graph. so add all of the ancestors of CURRENT to S.

104-366 WK 15 APR 2012

13 Friday

AO* Algorithm.

1. Initialize: Set $G^* = \{s\}$, $f(s) = h(s)$
   G^* = marked subtree (or subgraph) rooted at the start node s. if $s \in T$, label s as SOLVED.
   T = set of leaf or Terminal subproblems which are in it level subproblems, which can not be decomposed further but which have to be solved directly.

2. Terminate: if s is SOLVED, then Terminate.
   AND node is SOLVED when all children are SOLVED.
   OR node is SOLVED when the best cost child is SOLVED.

3. Select: Select the non-terminal leaf node n from the marked subtree.

4. Expand: Make explicit the children of n (Generate or expand the children of n).
   for each new child m:
   Set $f(m) = h(m)$
   if m is a terminal node, label m as SOLVED.

5. Cost-revision: Call cost-revise(n).

6. Loop: Go to step 2.

AO* Algorithm: Cost Revision

Procedure cost-revise(n)

1. Create Z={n}
   n= the node from where cost revision is to be started.
2. if Z={}, return.
3. Select a node m from Z such that m has no descendants in Z (because we want to go bottom up, we select a node which has no descendants).
4. if m is an AND node with children $r_1, r_2, \ldots, r_k$,
   Set $f(m) = \sum [f(r_i) + c(m, r_i)]$.
   where, $f(r_i)$ = Cost of child node,
   and, $c(m, r_i)$ = Cost of edge.
   mark the edge to each child of m with a double line indicating it is SOLVED. if each child is labeled SOLVED, then label m as SOLVED.
5. if m is an OR node with children $r_1, r_2, \ldots, r_k$,
   Set $f(m) = \min\{f(r_i) + c(m, r_i)\}$
   Mark the edge to the best child of m with a double line indicating it is SOLVED. if the marked child is labeled SOLVED, then label m as SOLVED.
6. if the cost or label of m has changed, then insert those parents of m into Z for which m is marked child.

Sunday 15

AO* always find a minimum cost solution if one exists. Also AO* is guaranteed to terminate even on graphs that have cycles.

107-366-WK 16
APR 2012

16 Monday

Game-Playing

Charles Babbage, the nineteenth-century computer architect, thought about programming his analytical engine to play chess and later of building a machine to play tic-tac-toe. Claude Shannon [1950] wrote a paper in which he described mechanisms that could be used in a program to play chess. A few years later, Alan Turing describes a chess playing program, although he never built it.

By the early 1960's, Arthur Samuel had succeeded in building the first significant, operational game-playing program. His program played checkers and in addition to simply playing the game, could learn from its mistakes and improve its performance.

There are two reason that game appeared to be a good domain in which to explore machine intelligence.

(i) They provide a structured task in which it is very easy to measure success or failure.

(ii) They did not require large amount of knowledge. They were thought to be soluble by straightforward search from the starting state to a winning position.

The first of these reasons remains valid and accounts for continued interest in the area of game playing by machine. Unfortunately, the second is not true for any but the simplest game. For eg. consider chess game.

→ The average branching factor is around 35.
→ In an average game, each player might makes 50 moves.
→ So In order to examine the complete game tree, we would have to examine 35^{100} positions.

Thus it is clear that a program that simply does a straightforward search of the game tree will not be able to select even its first move during the life-time of its opponent. Some kinds of heuristic search procedure is necessary. One way is to use the previous procedure. Generate-and-test procedures can be used in which testing is done after varying amounts of work by the generator. But to improve the effectiveness of a search-based problem solving program two things can be done:

(i) Improve the generate procedure so that only good moves (or paths) are generated.

(ii) Improve the test procedure so that the best moves (or paths) will be recognized and explored first.

Consider again the problem of playing chess. On the average, there are about 35 legal moves available at each turn. If we use simple legal-move generator, then the test procedure will have to look at each of them. So it must be fast.

Components of Game playing:-

(i) Plausible-move Generator

(ii) Static-evaluation function.

On the other hand, that instead of legal move generator, we use a plausible-move generator in which only small no. of promising moves are generated. As the no. of moves increases, it becomes increasingly imp to apply heuristic to select only these that have some kind of promise.

With a more selective move generator, the test procedure can afford to spend more time evaluating each of the moves it is given so it can produce a more reliable result. Thus by incorporating heuristic knowledge into both the generator and the tester, the performance of the overall system can be improved.

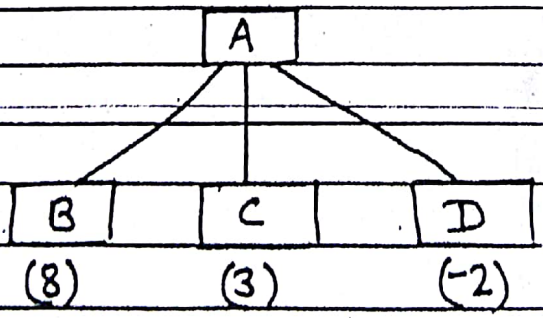
In order to choose best more, we choose static evaluation-functi which uses information to evaluate the individual path. It utilize the heuristic knowledge for evaluating the static evaluation function value.

THE MINMAX SEARCH PROCEDURE / MINMAX ALGORITHM

The minmax search procedure is a depth-first, depth limited search procedure. The idea is to start at the current position and use the plausible-move-generator to generate the set of possible successor positions. Now we can apply static evaluation function to those positions and simply the choose one the best one.

After doing so, we can back that value up to the starting position to represent our evaluation of it. Here we assume that the static evaluation function returns large values to indicating good situation for us, so our goal is to maximize the value of the static evaluation function of the next position.

Example:- Below diagram shows the operation

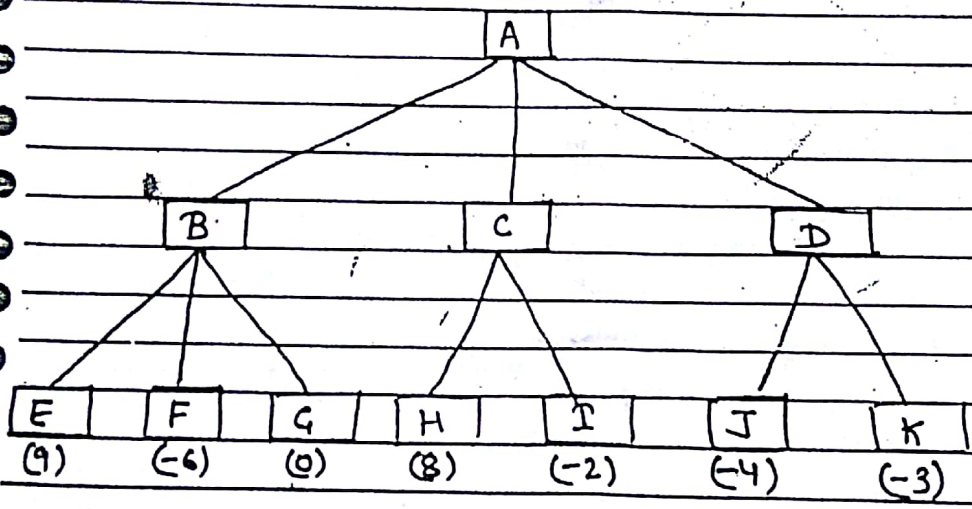


One-ply Search.

It assumes a static evaluation function that returns values ranging from -10 to 10, with 10 indicating a win for -10 a win for opponent, and 0 for even match.

Since our goal is to maximize the value of the heuristic function, we choose to move B. Backing B's value up to A, we can conclude that A's value is 8, so we can move to a position with a value of 8.

But we known that the static evaluation function is not completely accurate, we would like to carry the search further ahead than one ply. ~~For~~ for eg. in a chess game in which we are in the middle of a piece exchange. After our move, the situation would appear to be very good, but if we look one more ahead, we will see that one of our pieces also gets captured and so the situation is not favorable. So, we look one step ahead.

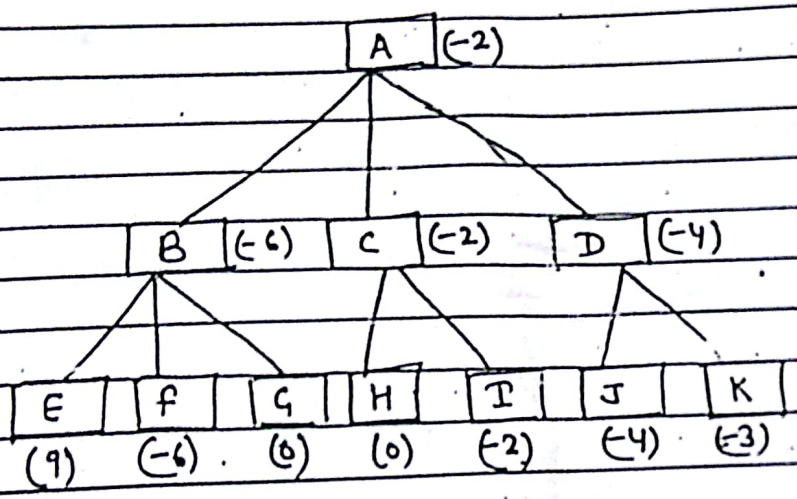


Two-ply Search

Thus, we apply the plausible-move generator, generating a set of successor positions for each position. If we wanted to stop here, at two-ply lookhead, we could apply the static evaluation function to each of these positions as shown above.

But now its turn of opponent. Suppose we made move B. then the opponent must choose among moves E, F, and G. The opponent's goal is to minimize the value of the evaluation function, so he or

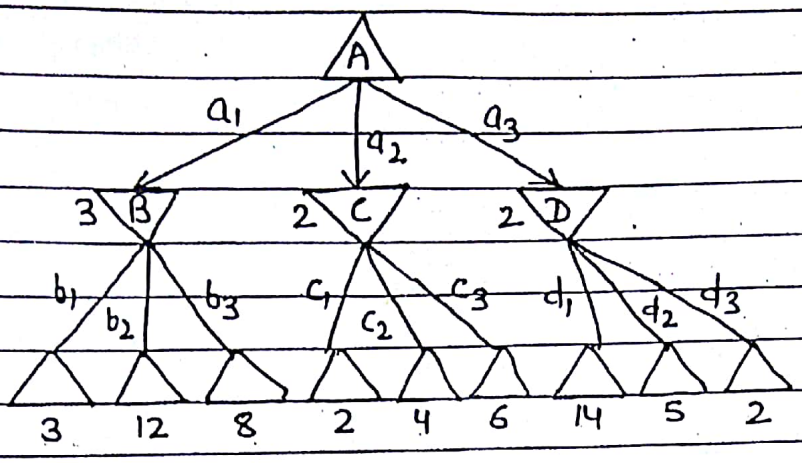
she can be expected to choose move F. This means that if we make move B, the actual position in which we will end up one move later is very bad for us. At this level, we are not the ones to move, we will not get to choose it.
Below fig. shows the result of propagating the new values up the tree. At that level, representing the opponent's choice, the minimum value was chosen and backed up.
At the level representing our choice, the maximum value was chosen.



Backing Up the Values of a Two-Ply Search.

Once the values from the second ply are backed up, we can choose the correct move for us to make at the first level is C, since there is nothing the opponent can do from there to produce a value worse than -2.
This process can be repeated for as many ply as time allows, and more accurate evaluations that are produced can be used to choose the correct move at the top level.

MINMAX



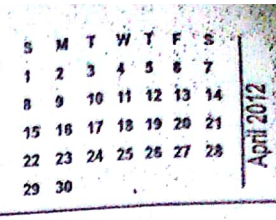
A two-ply game tree. The △ nodes are "MAX nodes" in which it is MAX's turn to move, and the ▽ nodes are "MIN nodes". The terminal nodes show the utility limit value for MAX; the other nodes are labeled with their minmax values. Max's best move at the root is a_1, because it leads to the successor with the highest minmax value, and MIN's best reply is b_1, because it leads to the successor with the lowest minmax value.

MIN-MAX Algorithm.

function MINMAX-DECISION (state) returns an action
  inputs: state, current state in game
    $v \leftarrow$ MAX-VALUE (state)
    return the action in SUCCESSORS (state) with value v.

function MAX-VALUE (state) returns a utility value
  if TERMINAL-TEST (state) then return UTILITY (state)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS (state) do
    $v \leftarrow$ MAX (v, MIN-VALUE (s))
  return v

Sunday 22

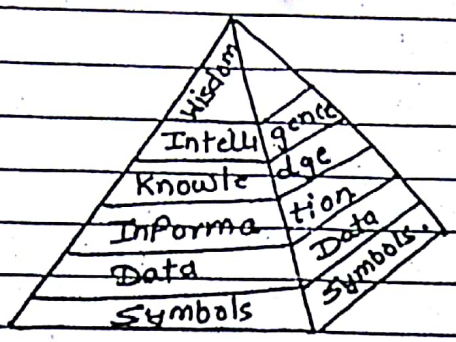



```
function Min-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

An algorithm for calculating minimax decisions. It returns the action corresponding to the best possible moves, i.e. the move that leads to the outcome with the best utility, under the assumption that opponent plays to minimize utility. The functions MAX-VALUE and MIN-VALUE go through the whole game-tree, all the way to the leaves, to determine the backed-up value of a state.

Knowledge.

The imp. role is that knowledge plays in building intelligent system is now widely accepted by practictioners in AI. The underlying thrust force behind every intelligent system is knowledge. fig. shows the knowledge pyramid.



Knowledge Pyramid.

At the base of the pyramid are symbols which form the means of representation. Data can be defined as a collection of mere symbols. when data are processed, one get information. Knowledge is organized information. Thus we can say, Knowled is a piece of info^{n} that helps in decision making. In biological organisms, Knowledge is likely stored as complex structure of interconnected neurons. But in computer, Knowledge is also stored as symbolic structures, but in the form of collecti of magnetic spots and voltage states.

A common way to represent knowledge exterinel to a computer or a human in the form of written language. for eg., some facts and relations represents in printed English are.

(i) Joe is tall.

Knowledge repn is a way of describing certain frames of information so that any reasoning system can easily adopt it for inferencing purpose.

KNOWLEDGE REPRESENTATION:-

SIGNIFICANCE OF KNOWLEDGE REPRESENTATION:-

Knowledge representation is a study of ways of how knowledge is actually pictuired and how effectively it resembles the representation of knowledge in human brain.

A Knowledge representation system should provide ways of representing complex knowledge and should possess the following characterstics:-

i) The representation sheme should have a set of well define syntax and semantics. This will help in representing various kind of knowledge.

(ii) The knowledge representation scheme should have a good expressive capacity. A good expressive capability will catalyze the inferencing mechanism in its reasoning process.

(iii) From the complex system point of view, the representation must be efficient.

REPRESENTATION OF KNOWLEDGE:-

Knowledge consists of facts, concepts, rules, and so forth. It can be represented in different forms, as mental images in one's thoughts, as spoken or written words in some language, as graphical or other pictures, and as character strings or collection of magnetic spots stored in a computer

P.T.O.

Saturday 05 MAY 2012

Mental images

↓

Written text

↓

Character strings

↓

Binary numbers

↓

Magnetic spots

Different levels of knowledge representation.

Any choice of representation will depend on the type of proble to be solved and the inference methods available.

CONTROVERSY OF PROCEDURAL V/S DECLARATIVE REPRESENTATION

This controversy raged in 1970's wherein there was heavy debate on which type of representation should be used in AI programs.

DECLARATIVE REPRESENTATION

A declarative representation declares every piece of knowledge and permits the reasoning system to use the rule of Sunday 06 inference like modus ponens, modus tolens etc. to come out with new piece of info^n. for e.g. consider the statements

p.t.o

"All carnivours have sharp teeth",
"Cheetah is a carnivoure".

This can be represented using a declaritive representation as:

$\forall x$ (carnivore (x) → sharp_teeth (x))
carnivore (cheetah).

Using these two representation, it is possible to deduce that "Cheetah has sharp teeth". This type of representation has the following advantages.

(1) Declaritive approaches are flexible.
(2) Each piece of knowledge is an independent chunk onits own.
(3) It is enough that you represent the knowledge only once

PROCEDURAL REPRESENTATION:-

A procedual representation, on the other hand, represents knowledge as procedure and inferencing mechanisms manipulate these procedures to auire at the result. for e.g.

```
procedue carnivore (X);
    if (X = cheetah) then return true
        else return false.
end procedue carnivore (X).

procedue sharp_teeth (X);
    if carnivore (X) then return true
    else return false.
end procedue sharp_teeth (X).
```

To see wheather cheetah has sharp teeth, one should activate procedure sharp-teeth with var.x instantiated to value cheetah. This procedure calls procedure carnivors(x). in turn with the value of (x = cheetah). Procedure carnivor returns a true value and so is procedure sharp-teeth.

Procedual representation also have many advantages.

(1) heuristic knowledge can be eailly represented which is vital.
(2) one has the control over seach which is not available in declarative knowledge representations.

Today it is agreed that a knowledge representation scheme should have both procedual and declarative scheme for effective organisation of the knowledge base.
There are some representation scheme
(i) Predicate Logic
(ii) Semantic Nets
(iii) Frames
(iv) Conceptual dependancy
(v) scripts.

130-366-WK 19 MAY 2012 09 Wednesday

REPRESENTATION AND MAPPINGS:

In order to solve the complex problems encountered in AI, one needs both a large amount of knowledge and some mechanism for manipulating that knowledge to create solution to new problems.

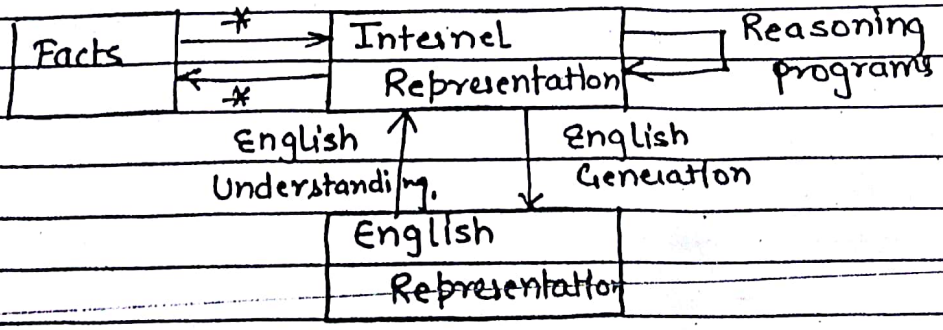
A variety of ways of representing knowledge (facts) have been exploited in AI programs.

There are two different kinds of activities entities:

(1) Facts: truths in some relevant world. These are the things we want to represent.

(2) Representation of facts in some chosen formalism. These are the things we will actually be able to manipulate.

Mapping between facts and Representation



There are two-way mapping for representation

(i) forward mapping.

(ii) Backward mapping.

forward mapping: It maps from facts to internel representation

Backward mapping: It maps from internel representation to facts

Thursday 10 MAY 2012

One representation of facts is so common that it deserves special mention: natural language sentences (particularly English).
A simple example:
Consider the English sentence.

Spot is a dog.

The above fact represented by that english sentence can also be represented in logic as

dog(spot)

Suppose that we also have a logical representation of the fact that all dogs have tails.

$\forall x : dog(x) \rightarrow hastail(x)$.

Then using the deductive mechanisms of logic, we may generate the new representation object:

~~hastail(spot)~~ hastail(spot)

Using an appropriate backward mapping function, we then generate the english sentence.

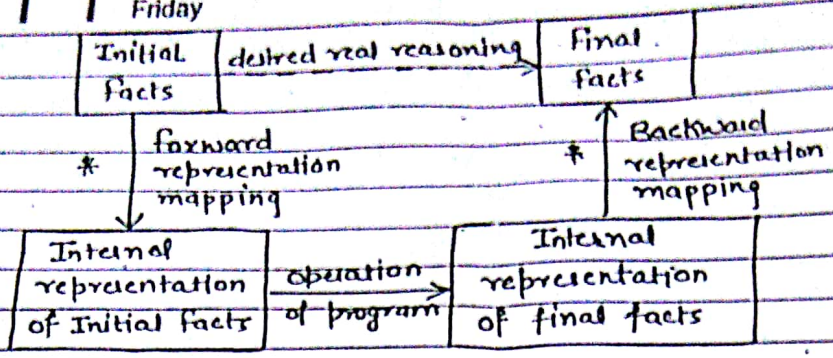
Spot has a tail.

Expanded - View of Mapping between facts and representation.

Below diagram shows an expanded-view of mapping b/w facts & representation

P.T.O.

MAY 2012 | 11 | Friday



Representation of facts.

The dotted line across the top represents the abstract reasoning process that a program is intended to model. The solid line across the bottom represents the concrete reasoning process that a particular program performs.

APPROACHES TO KNOWLEDGE REPRESENTATION

A good system for representation of knowledge has the following four properties

(i) Representation Adequecy - ability to represent all of kinds of knowledge that are needed in that domain.

(ii) Inferential Adequecy - ability to manipulate the representation structure in such a way as to derive new structures corresponding to new knowledge inferred from old.

(iii) Inferential Efficiency - ability to incorporate into the knowledge structure.

(iv) Acquisitional Efficiency - ability to acquire new info easily.

Saturday 12 MAY 2012

ISSUES IN KNOWLEDGE REPRESENTATION

(i) Are any attributes of objects so basic that they occur in almost every problem domain? If such attributes exist, what are they?

(ii) Are there any imp relationships that exist among attributes of objects?

(iii) At what level should knowledge be represented? Is there a good set of primitives into which all knowledge can be broken down?

(iv) How should set of objects be represented?

(v) Given a large amount of knowledge stored in a database, how can relevant parts be accessed when they are needed?.

(i) Important Attributes

There are two imp attributes that are of very general significance. instance and isa. These attributes are imp. because they support property Inheritance. They are called a variety of things in AI systems, but the names do not matter. The main use of these attributes represent class-membership and class inclusion.

(ii) Relationships among Attributes

The attributes have properties which is independent of the specific knowledge they endoce. These are four such properties.

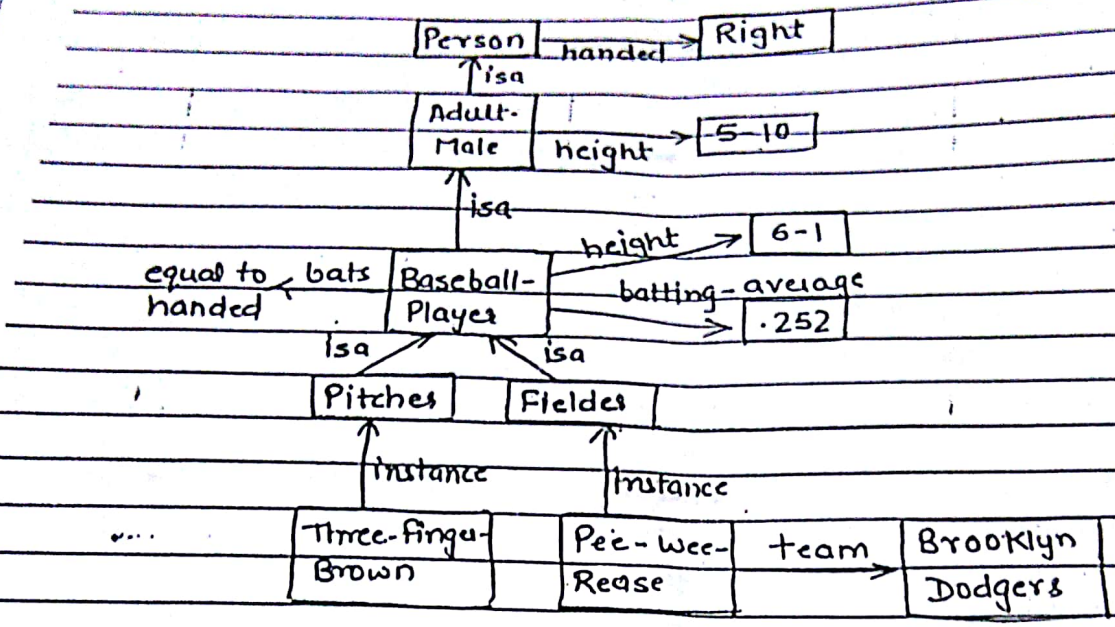
Sunday 13

(1) Inverses
(2) Existence in an isa hierarchy
(3) Techniques for reasoning about values
(4) Single-valued attributes

MAY 2012 | 14 Monday

Inverses.

Entities in the world are related to each other in many different ways. Attributes are relationships. For eg. in below fig. we used the attributes instance, isa and team.



Each of these is shown above by directed arrow, orginating at the object that was being described and terminating at the object representing the value.

There are two good ways to represent the view of relationship. The first is to represent both relationships in a single representation for eg

team (Pee-Wee-Reese, Brooklyn-Dodgers)

can equally easily be interpreted as a statement about Pee-wee-Reese or about the Brooklyn-Dodgers.

Tuesday 15 MAY 2012

The second approach is to use attributes that focus on a single entity but to use them in pairs, one the inverse of the other. In this approach we would represent the team information with two attributes:

- one associated with Pee Wee Reese:
  team = Brooklyn-Dodgers
- one associated with Brooklyn-Dodgers:
  team-members = Pee-Wee-Reese, ...

This is the approach that is taken in semantic net and frame-based systems. In this each time a value is added to one attribute that the corresponding value is added to the inverse.

An Isa Hierarchy of Attributes

Just as there are classes of objects and specialized subsets of those classes, there are attributes and specialization of attributes. Consider, for eg. the attribute height. This is specialization. These specialization-generalization relationships are imp. for attributes.

Techniques for reasoning about Values

Single-Valued Attributes

A specific but very useful kind of attribute is one that is guaranteed to take a unique value. For e.g, a baseball player can, at any one time, have only a single height and be a member of only one team.

137-366 WK 20 MAY 2012 16 Wednesday

(iii) Choosing the Granularity of Representation.

(iv) Representation set of objects.

(v) Finding the Right structures as Needed

Knowledge Representation

(1) Predicate logic
(2) Semantic nets.
(3) Frames
(4) Inheritance.

(1) Predicate Logic:-

Logic is a science of reasoning or arguments. or Logic is about reasoning. It is about the validity of arguments, consistency among statements and the matter of truth & falsehood.

Logic is a formal language for representing Info^n such that conclusion can be known.

Basically the logic process takes in some info^n called premises and produces some outputs (called conclusions). Logic is basically classified into two categories.

(i) Propositional logic
(ii) Predicate logic.

(i) Propositional logic

This is the simplest form of logic. Here all statements made are called proportions. A proportional in proportional logic takes only two values i.e. TRUE or it is FALSE.

037-365-WK 06 FEB 2012

06 Monday

PROLOG - THE NATURAL LANGUAGE OF ARTIFICIAL INTELLIGENCE

Prolog, which stands for PROgramming in LOGic, is the most widely available language in the logic programming area. The difference b/w Prolog and other languages is that a Prolog program tells the computer what to do (a technique called declarative programming) while programs in other languages tell the computer how to do it (procedural programming).

Prolog does this by making deductions and derivations from facts and rules stored in a database.

- Developed in 1972 by Alain Colmerauer and Philippe Roussel.
- Name comes from "PROgramming In LOGic"

Salient features of PROLOG:-

1. PROLOG is a declarative language. We encode the facts and relationships of the system under investigation, to make it part of KB of the system. The user can thus query for the system for information.

2. PROLOG uses the languages of predicate calculus.

3. PROLOG handles lists and recursions naturally.

4. PROLOG language has inbuilt inference engine and automatic backtracking facility.

5. Parallelism is inherent in PROLOG.

Tuesday 07 FEB 2012

PROBLEMS, PROBLEM SPACES AND SEARCH

Ques. What do you mean by problem state space?
Ques. How do you visualize an AI problem as state space? Explain Water-Jug problem.

To build a system to solve a particular problem, we need to do four things:

1. Define the problem precisely.
2. Analyze the problem.
3. Isolate and represent the task knowledge that is necessary to solve the problem.
4. Choose the best problem-solving technique(s) and apply it (them) to the particular problem.

Define the problem as a State Space Search

Problem State Space:-

The problem state space define the precise specification of what the initial situation will be as well as what final situations constitute acceptable solutions to the problem.

The state space representation forms the basis of most of the AI methods.

Example in Real life:-

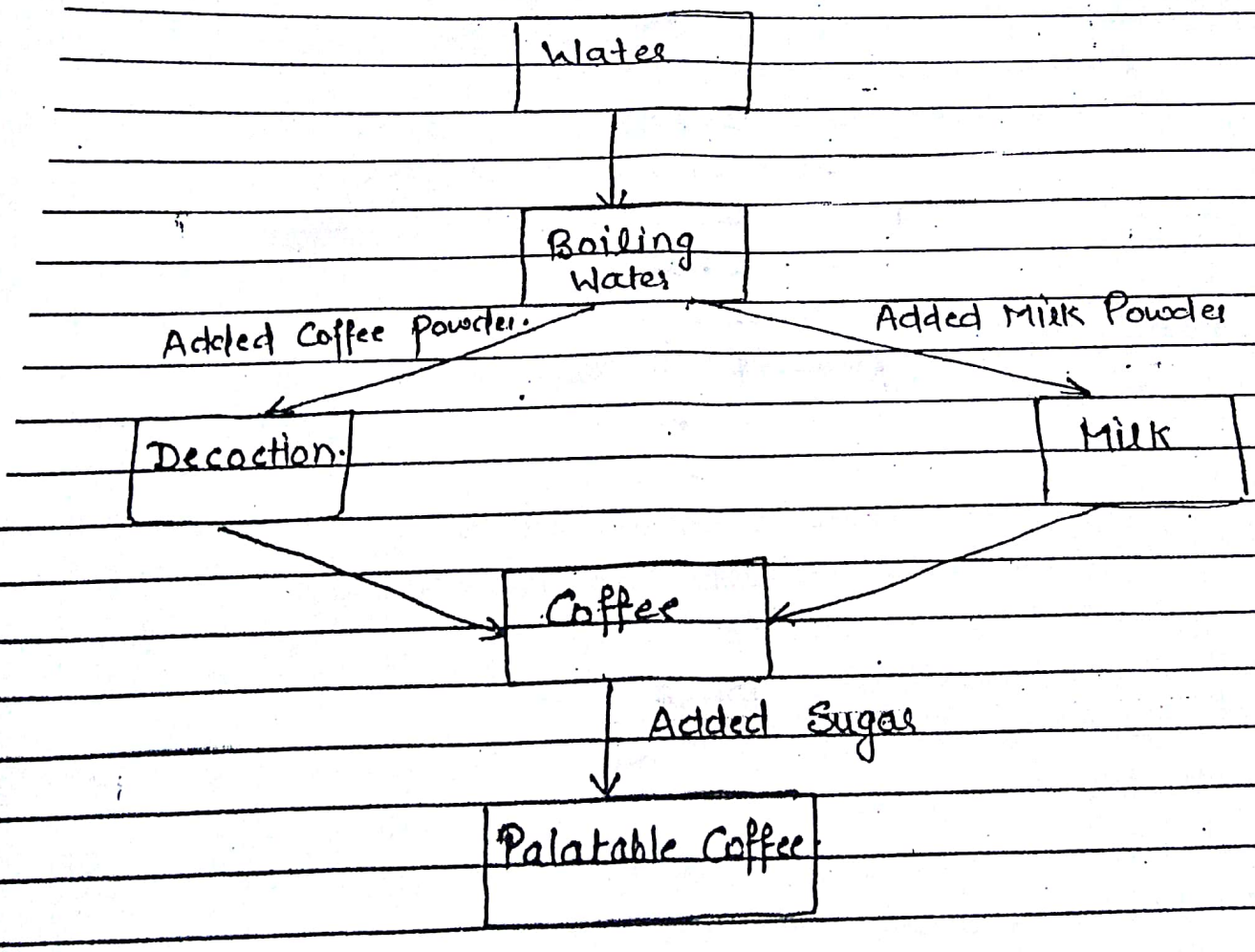
Suppose you are asked to make a cup of coffee. What will you d
You will verify wheather the neccessary ingredients like instant coffee powder, sugar, kettle, stove etc. are available.

If So, you will follow the following steps.

339-366-WK 06
FEB 2012
08 Wednesday

S M T W T F S
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29
February 2012

1. Boil necessary water in kettle.
2. Take some of the boiled water in a cup. add necessary amount of instant coffee powder to make decoction.
3. Add milk powder to the remaining boiling water to make coffee.
4. Mix decoction and milk.
5. Add sufficient quantity of sugar to your taste and coffee is ready.



State space representation for coffee making

Thursday 09 FEB 2012

In above example, you started with the ingredients (initial state), followed a sequence of steps (called states) and at last had a cup of coffee (goal state). You added only needed amount of coffee powder, milk powder and sugar (operators). Thus.

A set of all possible states for a given problem is known as the state space of the problem.

FIRST PROBLEM: Playing Chess

To build a program that could "Play Chess", we first have to specify the starting position of the chess board, the rules that define the legal moves, and the board positions that represent a win for one side or the other. for the problem "play chess", it is easy to provide a formal and complete description. The starting position can be described as an 8-by-8 array where each position contains a symbol standing for the appropriate piece.

We can define as our goal any board position in which the opponent does not have a legal move and his/her King is under attack. The legal move provide the way of getting from the initial state to a goal state.

They can be described easily as set of rules containing of two parts: a left side that serves as a pattern to be matched against the current board position and a right side that serves as a pad describe the change to be made to the board position to reflect the move.

Below shows Example to write the rule for a single move of a pawn of white.

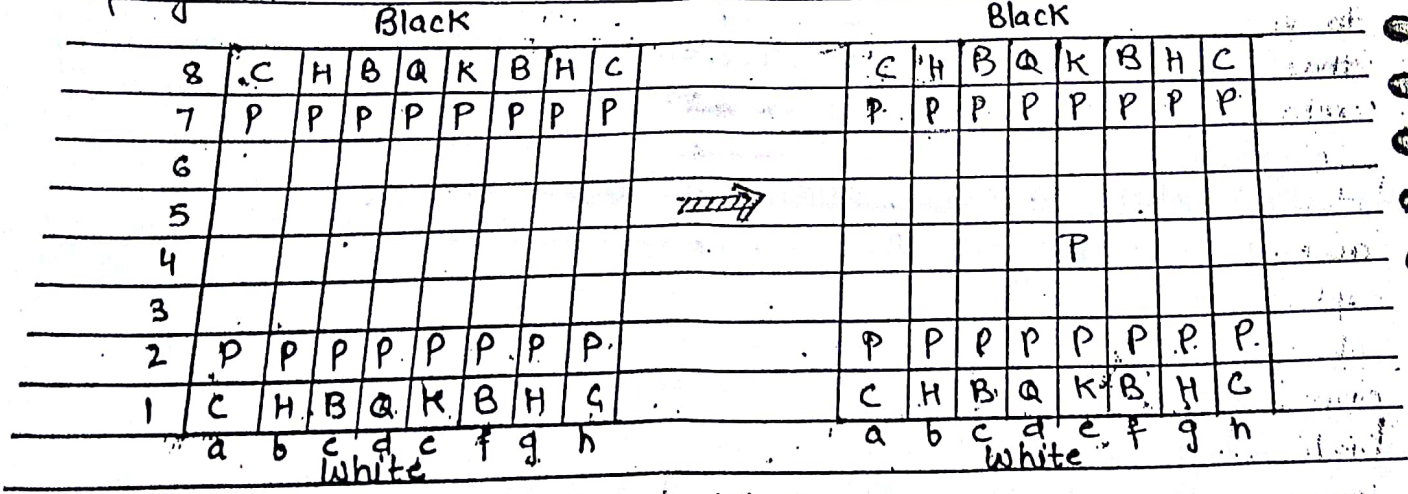
041-366-WK 06 FEB 2012

10 Friday

February 2012

White pawn at
Square (file e, rank 2)
AND
Square (file e, rank 3) → move pawn from
is empty Square (file e, rank 2)
AND to
Square (file e, rank 4) Square (file e, rank 4)
is empty.

Below diagram shows the representation of chess playing program.



One legal Chess Move

The problem of playing chess as a problem of moving around in a state space, where each state corresponding to a legal position of the board. We can then play chess by starting at an initial state, using set of rules to move from one state to another, and attempting to end up in one of a set of final states.
This state-space representation seems natural for chess.

because the set of states, which corresponds to the set of board positions, is artificial and well organized.

Second Problem: Water Jug Problem.
In this problem, we have given two jugs, one is 4-gallon and second is 3-gallon. Neither has any measuring markers on it. There is only one pump that can be used to fill the jug with water. How can we get exactly 2 gallons of water into 4 gallon jug?

Sol.
The state space of this problem can be describe as set of ordered pairs of integer (x,y).

$x = 0, 1, 2, 3, 4$

$y = 0, 1, 2, 3$

where x represents the no. of gallons of water in 4 gallon jug and y represents the 3 gallon jug water.

The start state is (0,0) and Goal state is (2,n) for any value of n.

We have assumed that we can fill a jug from pump, that we can pour water out of a jug onto the ground that we can pour water from one jug to another and there are no other measuring device available.

Some rules for solving the problem given below:-

1. $(x, y) \rightarrow (4, y)$ fill the 4gallon jug.
   if $x < 4$

2. $(x, y) \rightarrow (x, 3)$ fill the 3gallon jug.
   if $y < 3$

FEB 2012 13 Monday

3.	$(x,y) \rightarrow (x-d, y)$ if $x>0$	Pour some water out of 3gallon jug.
4.	$(x,y) \rightarrow (x, y-d)$ if $y>0$	pour some waters out of 4gallon jug.
5	$(x,y) \rightarrow (0, y)$ if $x>0$	Empty the 4gallon jug on the ground.
6.	$(x,y) \rightarrow (x, 0)$ if $y>0$	Empty the 3gallon jug on the ground.
7.	$(x,y) \rightarrow (4, y-(4-x))$ if $x+y \geq 4$ and $y>0$	Pour water from the 3gallon jug into the 4 gallon jug until the 4gallon jug is full.
8.	$(x,y) \rightarrow (x-(3-y), 3)$ if $x+y \geq 3$ and $x>0$	Pour water from the 4gallon into the 3 gallon jug until the 3 gallon jug is full.
9.	$(x,y) \rightarrow ((x+y), 0)$ if $x+y \leq 4$ and $y>0$	Pour all the waters from 3 gallon jug into the 4-gallon jug.
10.	$(x,y) \rightarrow (0, x+y)$ if $x+y \leq 3$ and $x>0$	Pour all water from 4 gallon jug into the 3-gallon jug.
11.	$(0,2) \rightarrow (2,0)$	Pour 2 gallon from 3gallon jug into 4 gallon jug.

12. (2,y) → (0,y) Empty the 2 gallon in the 4 gallon jug on the ground.

As rules whose left sides are matched against the current state and whose right sides describe the new state that result from applying the rule.

To solve the water jug problem, a control structure that loops through a simple cycle in which some rules whose left side matches the current state is chosen, appropriate change to the state is made as described in the corresponding right side and resulting state is checked to see if it corresponds to a goal state.

In order to provide a formal description of a problem, do following:-

1. Define a state space that contains all the possible configuration of the relevant object. It is possible to define this space without explicitly enumerating all of the states it contains.

2. Specify one or more states within that space describe possible situations from which of the problem solving process may start. These states are called initial state.

3. Specify one or more states that would be acceptable as solution to the problem. These states are called goal states.

4. Specify a set of rules that describe the action available. Doing this will require giving though to following issue:

FEB 2012 15 Wednesday

→ What unstated assumptions are present in the informal problem description?

→ How general should the rules be?

→ How much of the work required to solve the problem should be precomputed and represented in the rules?

Advantages and Disadvantages of state-space representations

Advantages:- This representation is very useful in AI because it provides a set of all possible states, operations and goals. If the entire state-space representation for a problem is given then it is possible to trace the path from the initial to goal state and identify the sequence of operators required for doing it.

Disadvantages:- It is not possible to visualize all the states for a given problem. Also, the resources of the computer system are limited to handle huge state-space representation.

Example 3. The eight tile puzzle problem formulation.

The eight tile puzzle consists of 3-by-3 square frame board which holds eight movable tiles numbered as 1 to 8. One square is empty, allowing the adjacent tiles to be shifted. The objective of the puzzle is to find a sequence of tile movements that leads from a starting configuration to goal configuration.

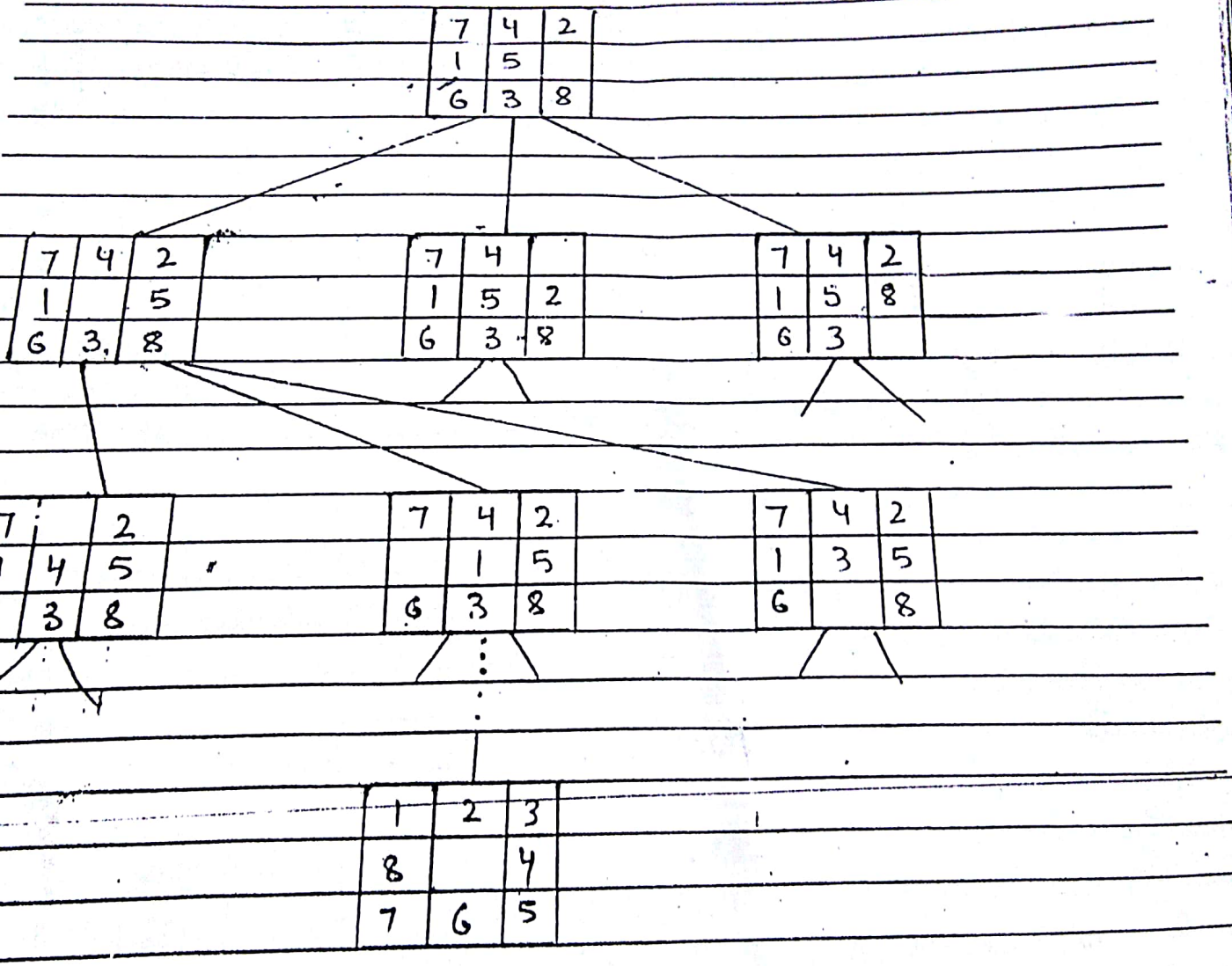
Thursday 16 FEB 2012

7	4	2
1	5	
6	3	8

to be transformed →

1	2	3
8		4
7	6	5

START STATE (initial) GOAL STATE (final)



Tree diagram (small part) of 8 puzzle problem.

FEB 2012 17 Friday

February 2012

No. of Solutions to water-jug Problem

Sol. 1.

Water in 4-Gallon jug	Water in 3-Gallon jug	Rule applied
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 or 12
2	0	9 or 11

Sol. 2.

Water in 4-Gallon jug	Water in 3-Gallon jug	Rule applied
0	0	
4	0	1
1	3	8
1	0	6
0	1	10
4	1	1
2	3	8

1 Bb Ba, A → B

February 2012

S	M	T	W	T	F	S
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29			

February 2012

Saturday 18 FEB 2012

Production Systems

These sym systems were proposed by Emil Post in 1943. They are also known a inferential systems, Rule based systems or simply productions.

Sunday 19