Artificial Intelligence

It is the study of how to make computers do things which, at the moment, people do better. It leads **FOUR** important categories.

i) Systems that think like humans
ii) ii) Systems that act like humans
iii) Systems that think rationally
iv) iv) Systems that act rationally

Acting humanly:

The Turing Test approach: To conduct this test, we need two people and the machine to be evaluated. One person plays the role of the interrogator, who is in a separate room from the compute and the other person. The interrogator can ask questions of either the person or the computer by typing questions and receiving typed responses. However, the interrogator knows them only as A and B and aims to determine which the person is and which are the machine. The goal of the machine is to fool the interrogator into believing that is the person. If the machine succeeds at this, then we will conclude that the machine is acting humanly. But programming a computer to pass the test provides plenty to work on, to posses the following capabilities.

Thinking Humanly:

The Cognitive modeling approach: To construct a machine program to think like a human, first it requires the knowledge about the actual workings of human mind. After completing the study about human mind it is possible to express the theory as a computer program. It the program‘s input/output and timing behavior matches with the human behavior then we can say that the program‘s mechanism is working like a human mind.
Example: General Problem Solver (GPS)

Thinking rationally:

The laws of thought approach: The right thinking introduced the concept of logic.
Example: Ram is a student of III year CSE. All students are good in III year in CSE. Ram is a good student

ACTING RATIONALLY:

Acting rationally means, to achieve one‘s goal given one‘s beliefs. In the previous topic laws of thought approach, correct inference is selected, conclusion is derived, but the agent acts on the conclusion defined the task of acting rationally.

AGENT

An **agent** is anything that can be viewed as **perceiving** its environment through **sensors** and **acting** upon that environment through **effectors**.
A **rational agent** is one that does the right thing.

We use the term **performance measure** for the *how*—the criteria that determine how MEASURE successful an agent is. Obviously, there is not one fixed measure suitable for all agents

agent program: a function that implements the agent mapping from percepts to actions. We assume this program will run on some sort of ARCHITECTURE(computing device) . Hence

A Agent is a combination of Architecture and Program.

agent = architecture + program

The Skeleton of an agent is

function SKELETON-AGENT(*percept*) **returns** action
static: *memory*, the agent's memory of the world
memory <-UPDATE-MEMORY(*memory*, *percept*)
action <-CHOOSE-BEST-ACTION(*memory*)
memory <-UPDATE-MEMORY(*memory*, *action*)
return *action*

A Table driven Agent function:

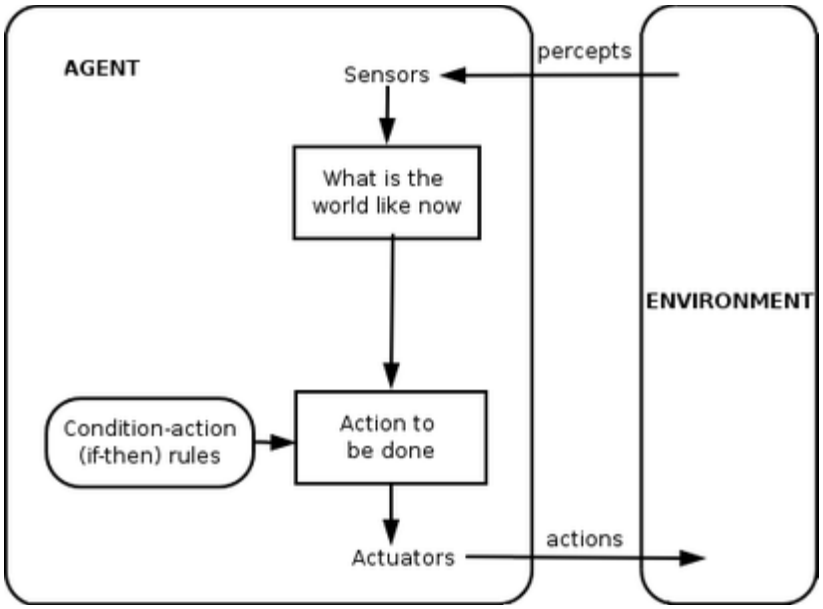
function TABLE-DRIVEN-AGENT(*percept*) **returns** *action*
static: *percepts*, a sequence, initially empty
table, a table, indexed by percept sequences, initially fully specified
append *percept* to the end of *percepts*
action <-LOOKUP(*percepts*, *table*)
return *action*

Four types of agent program:

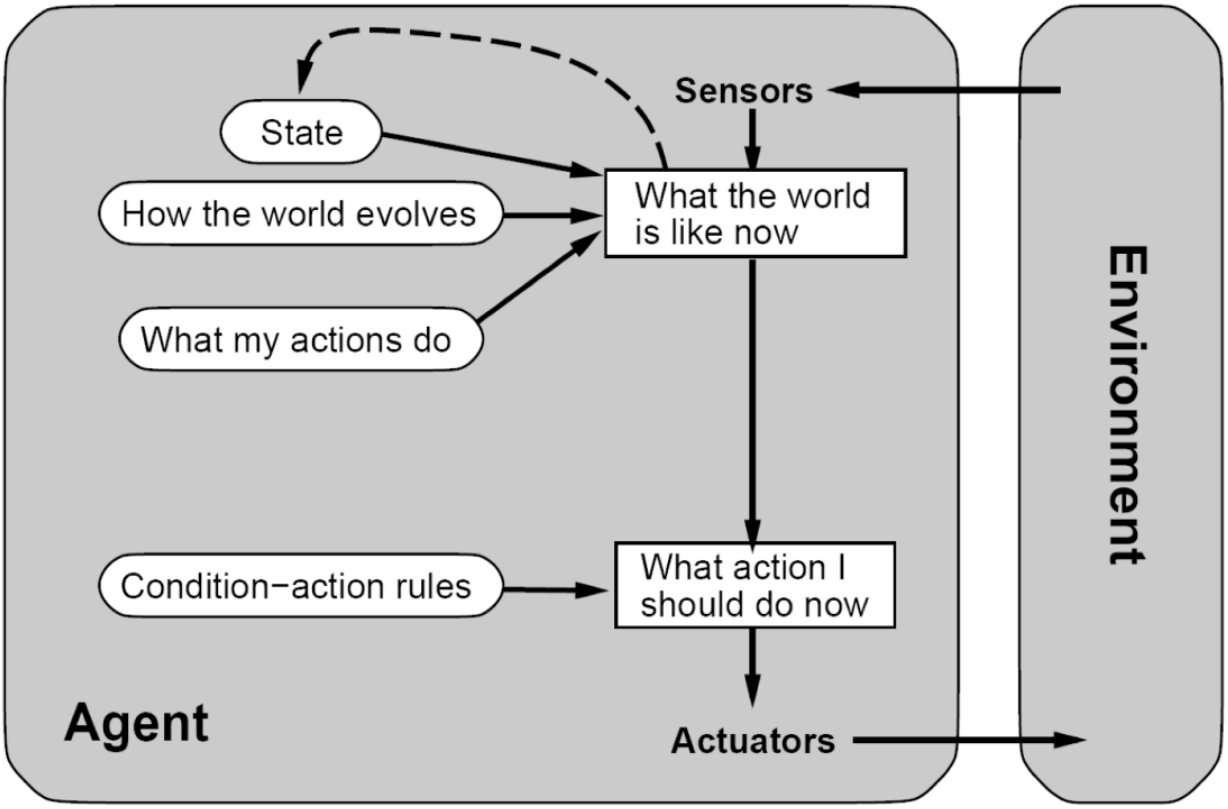
1. Simple reflex agents
2. Agents that keep track of the world
3. Goal-based agents
4. Utility-based agents

Simple reflex agents:

Structure of a simple reflex agent in schematic form, showing how the condition–action rules allow the agent to make the connection from percept to action.



function SIMPLE-REFLEX-AGENT(*percept*) **returns** *action*
static: *rules*, a set of condition-action rules
state <-INTERPRET-INPUT(*percept*)
rule <-RULE-MATCH(*state*, *rules*)
action <-RULE-ACTION[*rule*]
return *action*



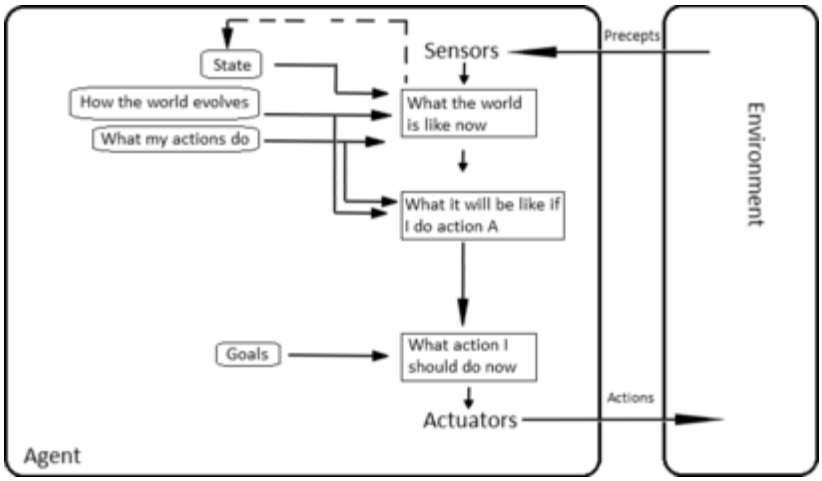
2.Agents that keep track of the world(Reflex Model)

The simple reflex agent described before will work only if the correct decision can be made on the basis of the current percept. In order to choose an action sometimes INTERNAL STATE will help to take good decision.

function REFLEX-AGENT-WITH-STATE(*percept*) **returns** *action*
static: *state*, a description of the current world state
rules, a set of condition-action rules
state <-UPDATE-STATE(*state*, *percept*)
rule <-RULE-MATCH(*state*, *rules*)
action <-RULE-ACTION[*rule*]
state <-UPDATE-STATE(*state*, *action*)
return *action*

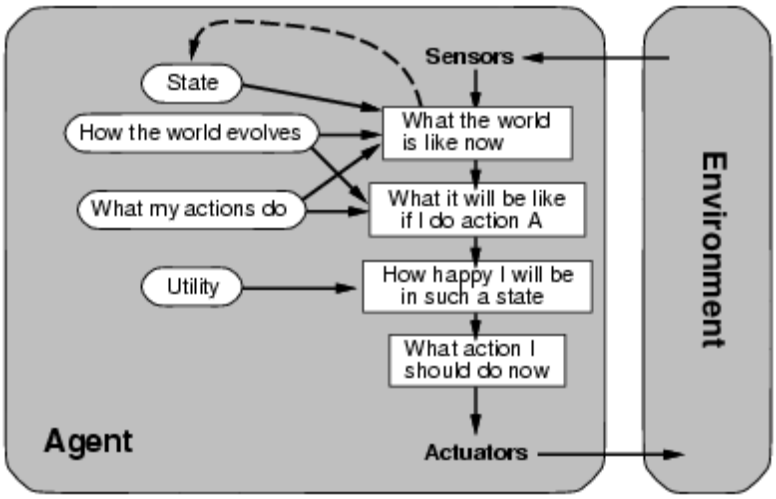
3. Goal-based agents

Knowing about the current state of the environment is not always enough to decide what to do. In other words, as well as a current state description, GOAL the agent needs some sort of **goal** information, which describes situations that are desirable.



4. Utility Based Model:

Goals alone are not really enough to generate high-quality behavior. For example, there are many action sequences that will get the taxi to its destination, thereby achieving the goal, but some are quicker, safer, more reliable, or cheaper than others. Goals just provide a crude distinction between “happy” and “unhappy” states, whereas a more general performance measure should allow a comparison of different world states. Then it has higher **utility** for the agent



Search:

Formulate a problem as a state space search by showing the legal problem states, the legal operators, and the initial and goal states .

• A state is defined by the specification of the values of all attributes of interest in the world

• An operator changes one state into the other; it has a precondition which is the value of certain attributes prior to the application of the operator, and a set of effects, which are the attributes altered by the operator

• The initial state is where you start

• The goal state is the partial description of the solution

State Space Search Notations

The set of notations involved in the state space search is :

1) An initial state is the description of the starting configuration of the agent

2) An action or an operator takes the agent from one state to another state which is called a successor state. A state can have a number of successor states.

3) A plan is a sequence of actions. The cost of a plan is referred to as the path cost.

Problem formulation & Problem Definition

Problem formulation means choosing a relevant set of states to consider, and a feasible set of operators for moving from one state to another.

Search is the process of considering various possible sequences of operators applied to the initial state, and finding out a sequence which culminates in a goal state.

A search problem consists of the following:

• S: the full set of states

• s0 : the initial state

• A:S→S is a set of operators

• G is the set of final states. Note that G ⊆S

The search problem is to find a sequence of actions which transforms the agent from the initial state to a goal state g∈G.

A simple search procedure is
Do until a solution is found or the state space is exhausted.
1. Check the current state
2. Execute allowable actions to find the successor states.
3. Pick one of the new states.
4. Check if the new state is a solution state
If it is not, the new state becomes the current state and the process is repeated
Basic Search Algorithm is
Let L be a list containing the initial state (L= the fringe)
Loop if L is empty return failure
Node <- select (L)
if Node is a goal
then return Node (the path from initial state to Node)
else generate all successors of Node, and merge the newly generated states into L
End Loop
Factors to measure the search Algorithms:
1. Complete
2. Optimal
3. What is the search cost associated with the time and memory required to find a solution?
a. Time complexity
b. Space complexity
The different search strategies that we will consider include the following:
1. Blind Search strategies or Uninformed search
a. Depth first search
b. Breadth first search
c. Iterative deepening search
d. Iterative broadening search
2. Informed Search
3. Constraint Satisfaction Search
Blind search:
Blind search or uninformed search that does not use any extra information about the problem domain. The two common methods of blind search are:
• BFS or Breadth First Search
• DFS or Depth First Search
Search Tree is helpful for the searching the goal node.
The terminologies involved in the search tree is
Root Node: The node from which the search starts.

Leaf Node: A node in the search tree having no children.

Ancestor/Descendant: X is an ancestor of Y is either X is Y's parent or X is an ancestor of the parent of Y. If S is an ancestor of Y, Y is said to be a descendant of X.

Branching factor: the maximum number of children of a non-leaf node in the search tree

Path: A path in the search tree is a complete path if it begins with the start node and ends with a goal node. Otherwise it is a partial path.

Breadth First Search

Let *Queue* be a list containing the initial state
Loop if *Queue* is empty return failure Node remove-first (fringe)
if Node is a goal
then return the path from initial state to Node
else generate all successors of Node, and
(merge the newly generated nodes into *fringe*)
add generated nodes to the back of *Queue*
End Loop
Advantages of Breadth First Search
Finds the path of minimal length to the goal.
Disadvantages of Breadth First Search
Requires the generation and storage of a tree whose size is exponential the the depth of the shallowest goal node

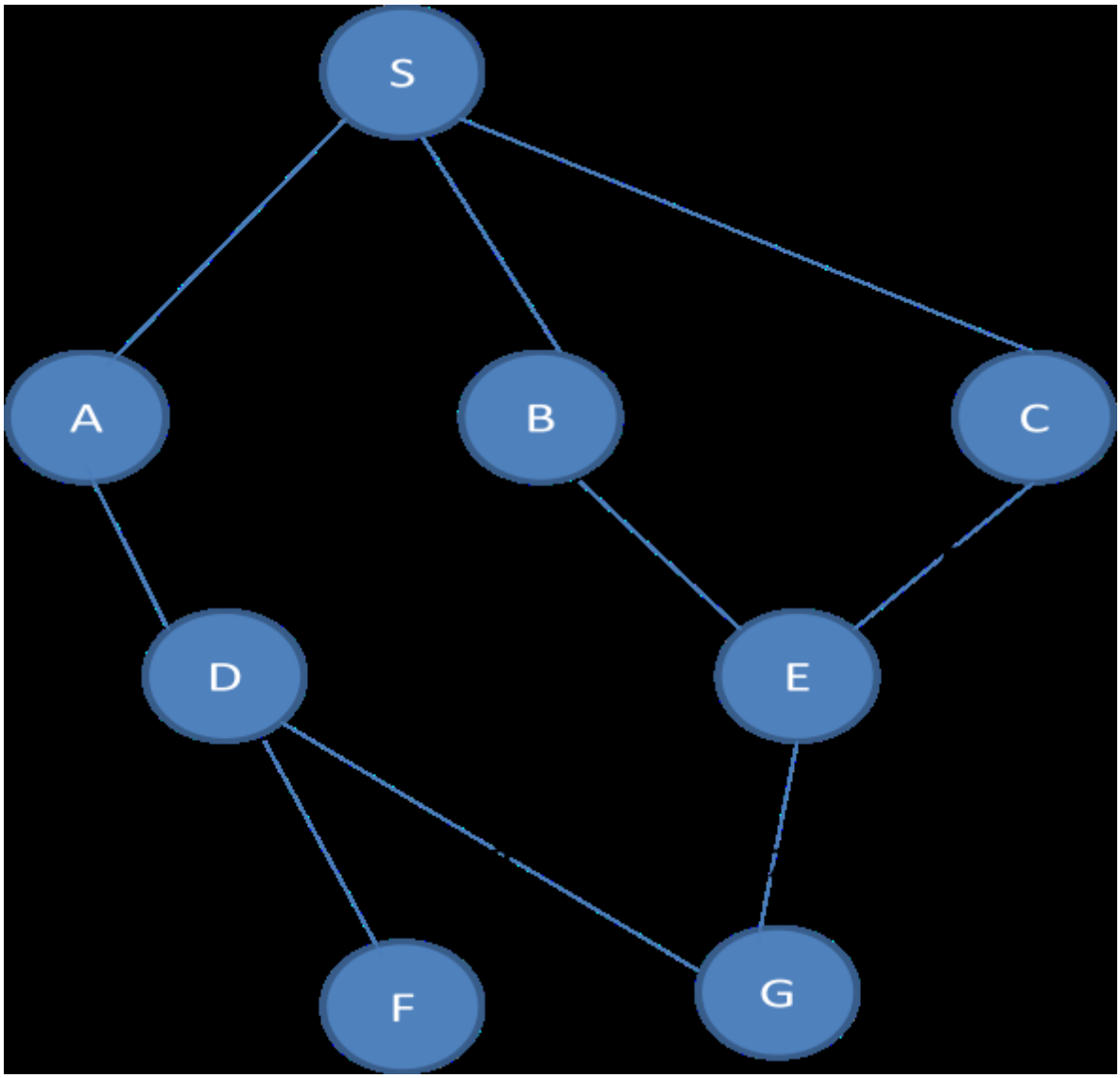
Depth First Search

Let *Queue* be a list containing the initial state
Loop if *Queue* is empty return failure
Node<-remove-first (*fringe*)
if Node is a goal
then return the path from initial state to Node
else generate all successors of Node,
and merge the newly generated nodes into *Queue*
add generated nodes to the front of *Queue*
End Loop
Depth Limited Search
Let fringe be a list containing the initial state
Loop if fringe is empty return failure
Node<-remove-first (fringe)
if Node is a goal
then return the path from initial state to Node
else if depth of Node = limit return cutoff
else add generated nodes to the front of fringe
End Loop
Depth-First Iterative Deepening (DFID)
until solution found do
DFS with depth cutoff c

c = c+1
A* algorithm.
f(n) = g(n) + h(n)
where *g(n)* = sum of edge costs from start to n
h(n) = estimate of lowest cost path from
f(n) = actual distan
h(n) is said to be admissible if it underestimates the cost reached from *n*.
If *C*(n)* is the cost of the cheapest
h(n) <= C(n)*. we can prove that if *h(n)* is admissible, then the search will find an optimal solution.

A* Algorithm:

OPEN = nodes on frontier.
CLOSED = expanded nodes.
OPEN={<S,NIL>} while OPEN is not empty place *<n,p>*
REMOVE FROM OPEN NODE<n,p>with minimum f(n)
if *n* is a goal node,
return success (path p)
for each edge connecting n & M with cost c
if *<m, q>* is on CLOSED and *{p|e}* is cheaper than *q*
then remove *n* from CLOSED,
put *<m,{p|e}>* on OPEN
else if *<m,q>* is on OPEN and *{p|e}* is cheaper than *q*
then replace *q* with *{p|e}*
else if *m* is not on OPEN
then put *<m,{p|e}>* on OPEN
return failure



Here the Starting node is S and the goal node is represented as G. Our aim is to reach the goal node G by tracing out the algorithm described

Status of OPEN and CLOSED list after each step:

NOTE: OPEN list and CLOSED list will be abbreviated as OL and CL from henceforth.

Whenever a node enters the OL,the value of its parent node and its total cost from the starting node S will be included within simple braces().

Step 1:

OL: S (NULL ; 0) CL: (NULL)

Step 2:

OL: A (S,1) , B(S,3) , C(S,10) CL:S

Step 3:

OL: B(S,3), C(S,10), D(A,6) CL: S,A

Step 4:
OL:C(S,10), D(A,6), E(B,7) CL: S,A,B
Step 5:
OL: D(A,6), E(B,7) CL: S,A,B,C
Step 6:
OL: E(B,7), F(D,8) CL: S,A,B,C,D
Step 7:
OL: F(D,8),G(E,14) CL: S,A,B,C,D,E
Step 8:
OL: G(E,14) CL: S,A,B,C,D,E,F
Step 9:
OL: -- CL: S,A,B,C,D,E,F,
Constraint satisfaction problems or **CSP**s are mathematical problems where one must find states or objects that satisfy a number of *constraints* or criteria. A constraint is a restriction of the feasible solutions in an optimization problem.
A **Constraint Satisfaction Problem** (CSP) is characterized by:
a *set of variables* {x1, x2, .., xn},

for each variable xi a *domain* Di with the possible values for that variable, and

a set of *constraints*, i.e. relations, that are assumed to hold between the values of the variables. [These relations can be given intentionally, i.e. as a formula, or extensionally, i.e. as a set, or procedurally, i.e. with an appropriate generating or recognising function.] We will only consider constraints involving one or two variables.

The constraint satisfaction problem is to find, for each i from 1 to n, a value in Di for xi so that all constraints are satisfied.
A CSP can easily be stated as a sentence in first order logic, of the form:
(exist x1)..(exist xn) (D1(x1) & .. Dn(xn) => C1..Cm)
• What is a CSP?
– Finite set of variables *V1, V2, …, Vn*
• Nonempty domain of possible values for each variable *DV1, DV2, … DVn*
– Finite set of constraints *C1, C2, …, Cm*
• Each constraint *Ci* limits the values that variables can take,
– e.g., $V1 \neq V2$
– A *state* is defined as an *assignment* of values to some or all variables.
• *Consistent assignment*
– assignment does not violate the constraints

– CSP benefits

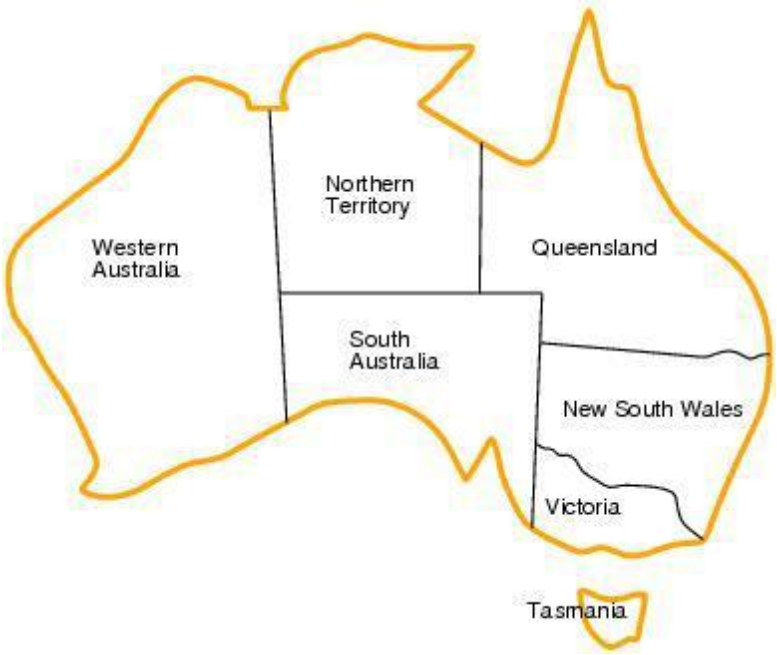
– Standard representation pattern

– Generic goal and successor functions

– Generic heuristics (no domain specific expertise).
• An assignment is *complete* when every variable is mentioned.

• A *solution* to a CSP is a complete assignment that satisfies all constraints.

Some CSPs require a solution that maximizes an *objective function*

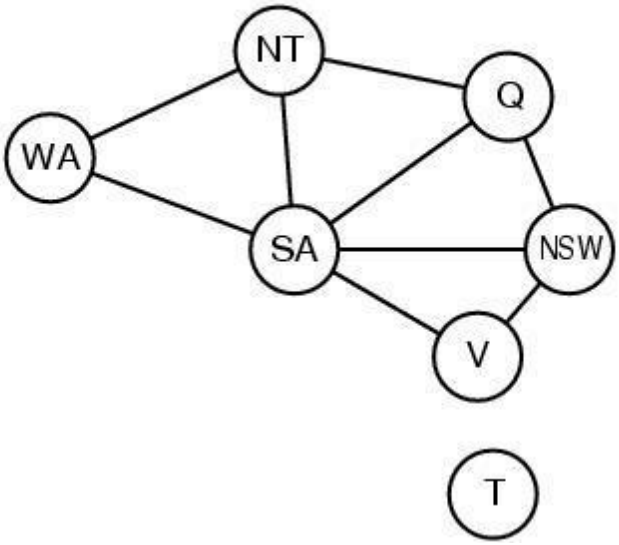


Variables: *WA, NT, Q, NSW, V, SA, T*

• Domains: *Di={red,green,blue}*

• Constraints:adjacent regions must have different colors.
• E.g. *WA NT*
• Solutions are assignments satisfying all constraints, e.g.
{WA=red,NT=green,Q=red,NSW=green,V=red,SA=blue,T=green}
• Constraint graph:
• nodes are variables

• arcs are binary constraints



• Graph can be used to simplify search

e.g. Tasmania is an independent subproblem

Types Of Variables:

Discrete variables

– Finite domains; size *d O(dn)* complete assignments.

• E.g. Boolean CSPs: Boolean satisfiability (NP-complete).

– Infinite domains (integers, strings, etc.)

• E.g. job scheduling, variables are start/end days for each job

• Need a constraint language e.g $StartJob1 + 5 \leq StartJob3$.

• Infinitely many solutions

• Linear constraints: solvable

• Nonlinear: no general algorithm

Continuous variables

– e.g. building an airline schedule or class schedule.

– Linear constraints solvable in polynomial time by LP methods.

Unary constraints involve a single variable.

– e.g. *SA green*

Binary constraints involve pairs of variables.

– e.g. *SA WA*

CSP as SEARCH:

• A CSP can easily be expressed as a standard search problem.

• Incremental formulation

– *Initial State*: the empty assignment {}

– *Successor function*: Assign a value to any unassigned variable provided that it does not violate a constraint

– *Goal test*: the current assignment is complete

– *Path cost*: constant cost for every step

CSP By BackTracking :

• Similar to Depth-first search

• Chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.

• Uninformed algorithm **function** BACKTRACKING-SEARCH(*csp*) **return** a solution or failure

return RECURSIVE-BACKTRACKING(*{} , csp*)

function RECURSIVE-BACKTRACKING(*assignment, csp*) **return** a solution or failure

if *assignment* is complete **then return** *assignment*

var SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)

for each *value* **in** ORDER-DOMAIN-VALUES(*var, assignment, csp*) **do**

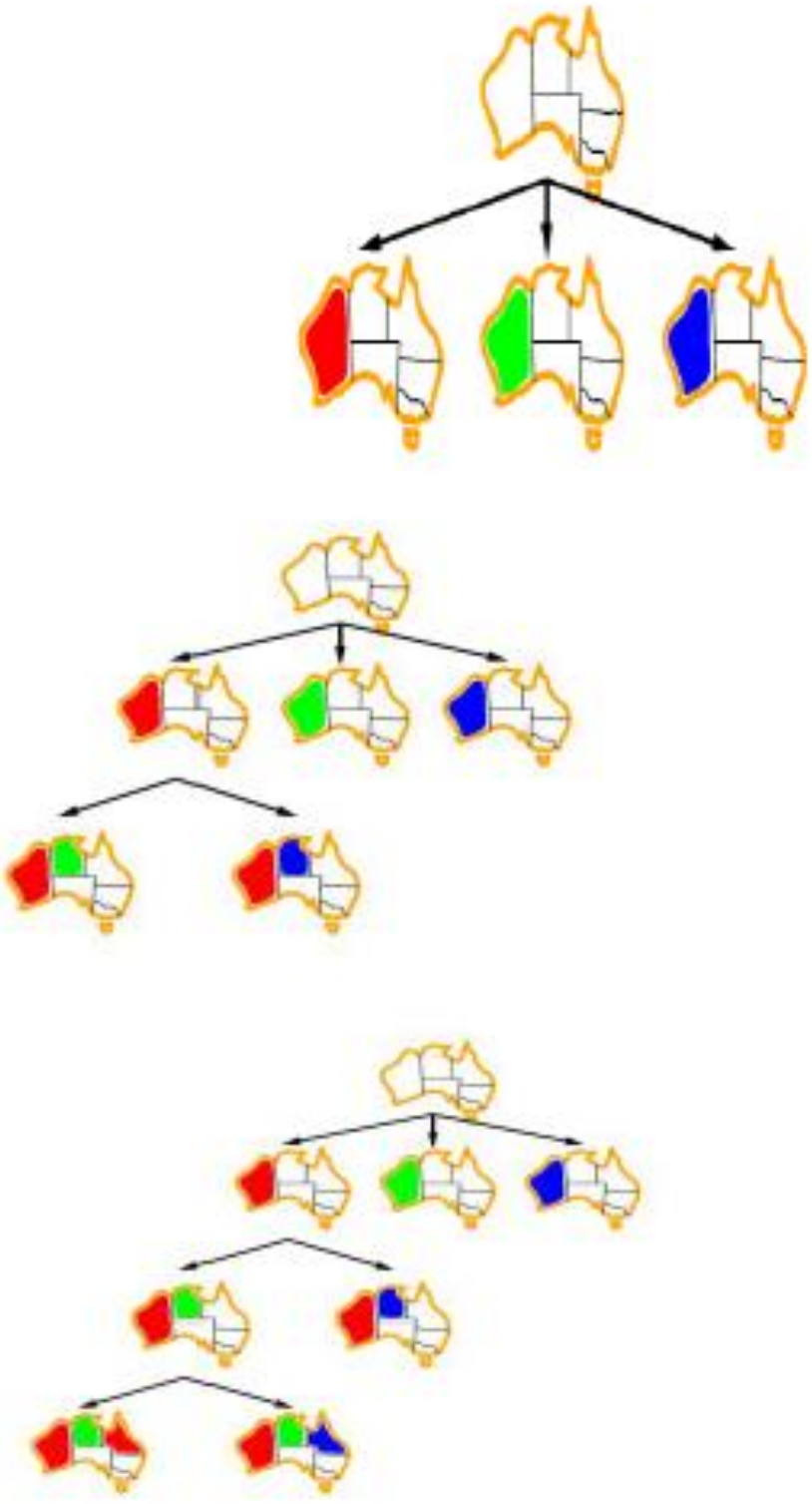
if *value* is consistent with *assignment* according to CONSTRAINTS[*csp*] **then** add *{var=value}* to assignment

result RRECURSIVE-BACTRACKING(*assignment, csp*)

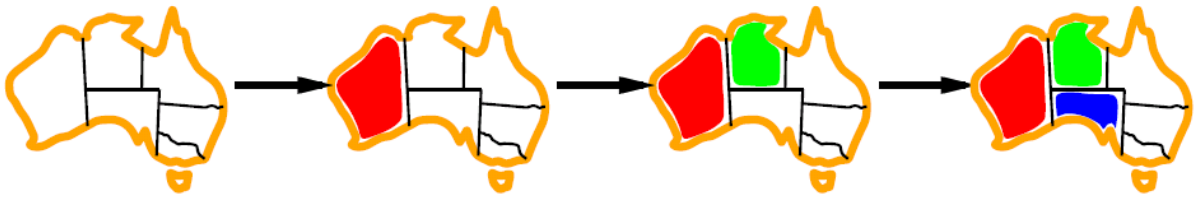
if *result failure* **then return** *result*

remove *{var=value}* from *assignment*

return *failure*



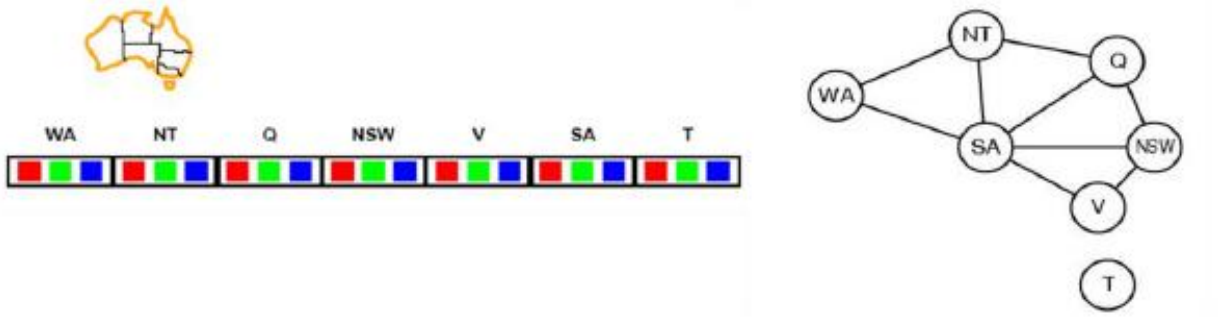
Minimum Remaining Values:



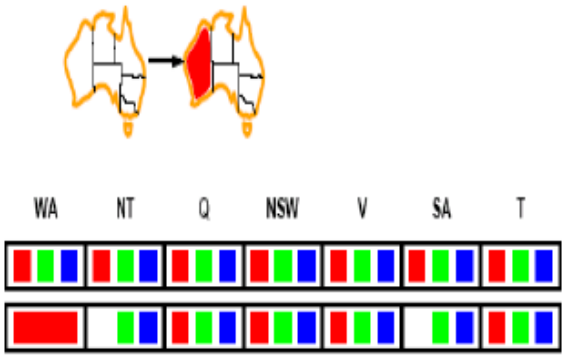
var SELECT-UNASSIGNED-VARIABLE(VARIABLES[*csp*],*assignment*,*csp*)
• A.k.a. most constrained variable heuristic

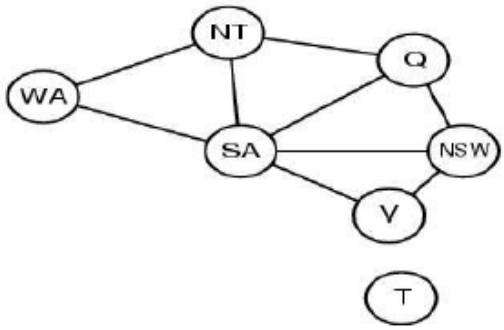
• *Heuristic Rule*: choose variable with the fewest legal moves
– e.g., will immediately detect failure if X has no legal values

Forward Checking:

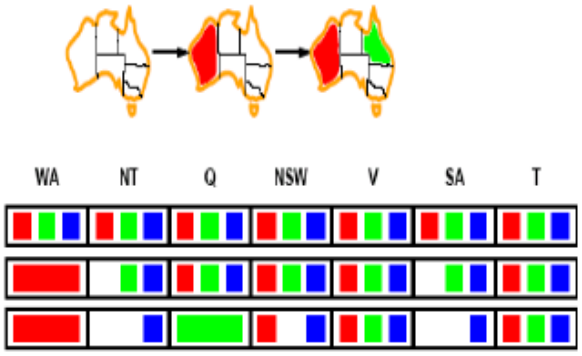


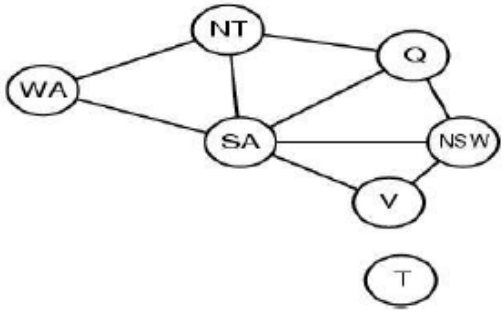
- Can we detect inevitable failure early?
  - *And avoid it later?*
- *Forward checking idea:* keep track of remaining legal values for unassigned variables.
- Terminate search when any variable has no legal values.

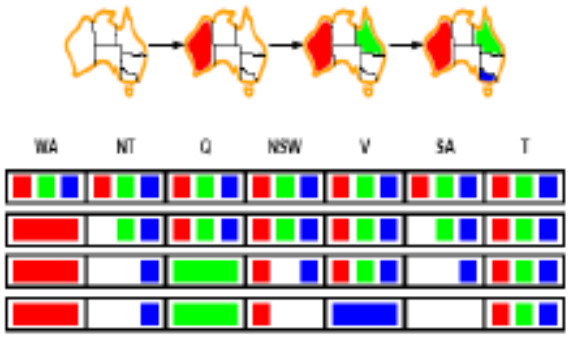


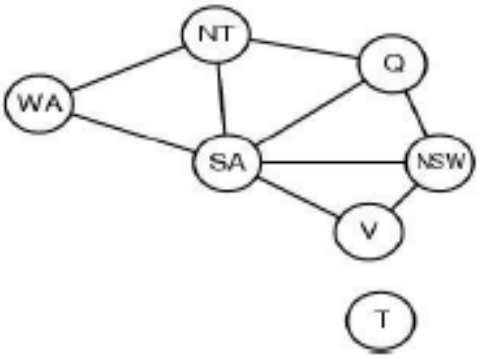


- Assign *{WA=red}*
- Effects on other variables connected by constraints to WA
  - *NT can no longer be red*
  - *SA can no longer be red*



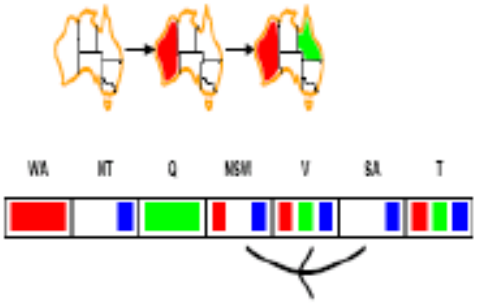


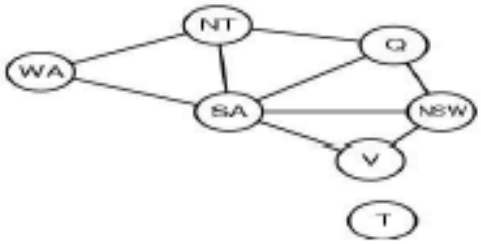




- If *V* is assigned *blue*
- Effects on other variables connected by constraints with WA
  - *NSW can no longer be blue*
  - *SA is empty*
  - FC has detected that partial assignment is *inconsistent* with the constraints and backtracking can occur.

Arc Consistency:





- *An Arc* $X \rightarrow Y$ is consistent if

  for *every* value x of X there is some value y consistent with x

(note that this is a directed property)

- Consider state of search after WA and Q are assigned:

$SA \rightarrow NSW$ is consistent if

REPRESENTATION OF KNOWLEDGE

Game playing - Knowledge representation, Knowledge representation using Predicate logic, Introduction to predicate calculus, Resolution, Use of predicate calculus, Knowledge representation using other logic-Structured representation of knowledge.

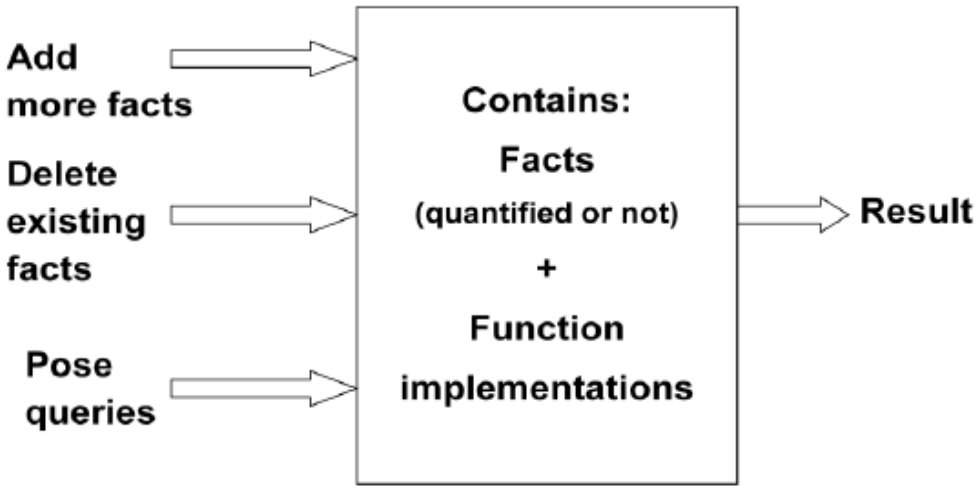
Knowledge Representation (KR)

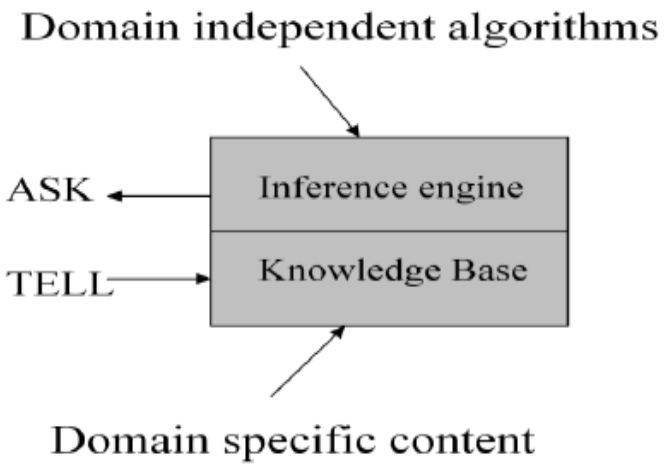
Given the world

- Express the general facts or beliefs using a language
- Determine what else we should (not) believe

Logic consists of

- A language
  which tells us how to build up sentences in the language (i.e., *syntax*), and and what the sentences mean (i.e., *semantics*)
- An inference procedure
  which tells us which sentences are valid inferences from other sentences





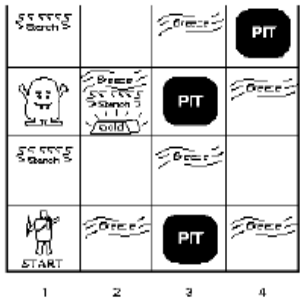

```
function KB-AGENT( percept) returns an action
   static: KB, a knowledge base
           t, a counter, initially 0, indicating time
   TELL(KB, MAKE-PERCEPT-SENTENCE( percept, t))
   action ← ASK(KB, MAKE-ACTION-QUERY(t))
   TELL(KB, MAKE-ACTION-SENTENCE( action, t))
   t ← t + 1
   return action
```

Wumpus World Problem:

Percepts Breeze, Glitter, Smell

Actions Left turn, Right turn,
Forward, Grab, Release, Shoot

Goals Get gold back to start
without entering pit or wumpus square



Environment

Squares adjacent to wumpus are smelly
Squares adjacent to pit are breezy
Glitter if and only if gold is in the same square
Shooting kills the wumpus if you are facing it
Shooting uses up the only arrow
Grabbing picks up the gold if in the same square
Releasing drops the gold in the same square

Characteristics of Wumpus World:

- Deterministic? Yes – outcome exactly specified.
- Accessible? No – only local perception.
- Static? Yes – Wumpus and pits do not move.
- Discrete? Yes
- Episodic? (Yes) – because static.

Logics are formal languages for representing information such that conclusions can be drawn

Syntax defines the sentences in the language

Semantics define the "meaning" of sentences; i.e., define truth of a sentence in a world

E.g., the language of arithmetic

$x+2 \geq y$ is a sentence; $x2+y >$ is not a sentence

$x+2 \geq y$ is true iff the number $x+2$ is no less than the number y

$x+2 \geq y$ is true in a world where $x=7,\ y=1$
$x+2 \geq y$ is false in a world where $x=0,\ y=6$

Propositional logic

The symbols of propositional calculus are

The propositional symbols:
P, Q, R, S, …

The truth symbols:
true, false

and connectives:
∧, ∨, ¬, →, ≡

Every propositional symbol and truth symbol is a *sentence*.
Examples: true, P, Q, R.

The *negation* of a sentence is a sentence.
Examples: ¬P, ¬ false.

The *conjunction*, or *and*, of two sentences is a sentence.
Example: P ∧ ¬P

The *disjunction*, or *or*, of two sentences is a sentence.
Example: P ∨ ¬P

The *implication* of one sentence from another is a sentence.

Example: P → Q

The *equivalence* of two sentences is a sentence.
Example: P ∨ Q ≡ R

Legal sentences are also called well-formed formulas or *WFF*s.

An *interpretation* of a set of propositions is the assignment of a truth value, either T or F to each propositional symbol.

The symbol true is always assigned T, and the symbol false is assigned F.

The truth assignment of *negation*, ¬P, where P is any propositional symbol, is F if the assignment to P is T, and is T is the assignment to P is F.

The truth assignment of *conjunction*, ∧, is T only when both conjuncts have truth value T; otherwise it is F.

Other approaches to inference use syntactic operations on sentences, often expressed in standardized forms

Conjunctive Normal Form (CNF—universal)

conjunction of $\underbrace{\textit{disjunctions of literals}}_{\textit{clauses}}$

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Disjunctive Normal Form (DNF—universal)

disjunction of $\underbrace{\textit{conjunctions of literals}}_{\textit{terms}}$

E.g., $(A \wedge B) \vee (A \wedge \neg C) \vee (A \wedge \neg D) \vee (\neg B \wedge \neg C) \vee (\neg B \wedge \neg D)$

Horn Form (restricted)

conjunction of *Horn clauses* (clauses with ≤ 1 positive literal)

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

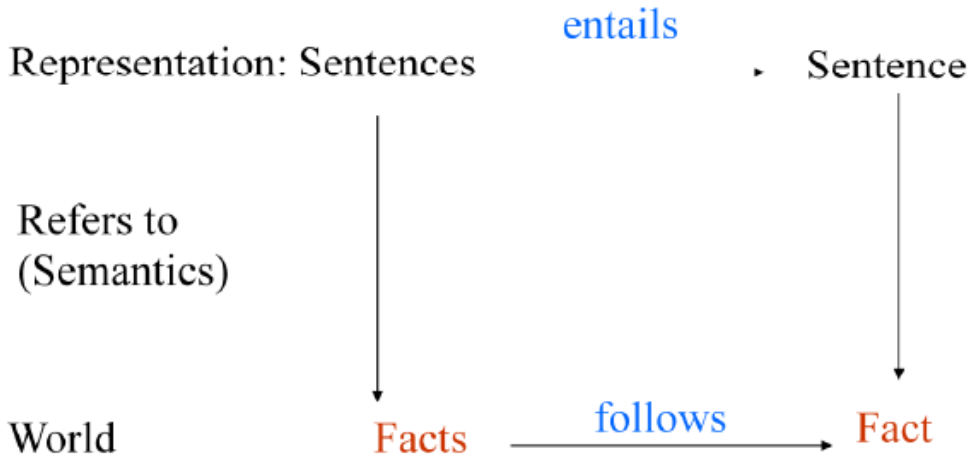
Often written as set of implications:

$B \Rightarrow A$ and $(C \wedge D) \Rightarrow B$

Entailment:

KB|= Sen1

The knowledge Base "KB" entails a Sentence "sen1" if and only if sen1 is true in the domain where the KB is true.



CNF :

Other approaches to inference use syntactic operations on sentences, often expressed in standardized forms

Conjunctive Normal Form (CNF—universal)

conjunction of *disjunctions of literals* (*clauses*)

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Disjunctive Normal Form (DNF—universal)

disjunction of *conjunctions of literals* (*terms*)

E.g., $(A \wedge B) \vee (A \wedge \neg C) \vee (A \wedge \neg D) \vee (\neg B \wedge \neg C) \vee (\neg B \wedge \neg D)$

Horn Form (restricted)

conjunction of *Horn clauses* (clauses with ≤ 1 positive literal)

E.g., $(A \vee \neg B) \wedge (B \vee \neg C \vee \neg D)$

Often written as set of implications:

$B \Rightarrow A$ and $(C \wedge D) \Rightarrow B$

◊ **Modus Ponens** or **Implication-Elimination**: (From an implication and the premise of the implication, you can infer the conclusion.)

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

◊ **And-Elimination**: (From a conjunction, you can infer any of the conjuncts.)

$$\frac{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}{\alpha_i}$$

◊ **And-Introduction**: (From a list of sentences, you can infer their conjunction.)

$$\frac{\alpha_1, \alpha_2, \ldots, \alpha_n}{\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n}$$

◊ **Or-Introduction**: (From a sentence, you can infer its disjunction with anything else at all.)

$$\frac{\alpha_i}{\alpha_1 \vee \alpha_2 \vee \ldots \vee \alpha_n}$$

◊ **Double-Negation Elimination**: (From a doubly negated sentence, you can infer a positive sentence.)

$$\frac{\neg\neg\alpha}{\alpha}$$

◊ **Unit Resolution**: (From a disjunction, if one of the disjuncts is false, then you can infer the other one is true.)

$$\frac{\alpha \vee \beta, \quad \neg\beta}{\alpha}$$

◊ **Resolution**: (This is the most difficult. Because β cannot be both true and false, one of the other disjuncts must be true in one of the premises. Or equivalently, implication is transitive.)

$$\frac{\alpha \vee \beta, \quad \neg\beta \vee \gamma}{\alpha \vee \gamma} \quad \text{or equivalently} \quad \frac{\neg\alpha \Rightarrow \beta, \quad \beta \Rightarrow \gamma}{\neg\alpha \Rightarrow \gamma}$$

Resolution

- Resolution is a sound and complete inference procedure
- Reminder: Resolution rule for propositional logic:
  - $P_1 \vee P_2 \vee \ldots \vee P_n$
  - $\neg P_1 \vee Q_2 \vee \ldots \vee Q_m$
  - Resolvent: $P_2 \vee \ldots \vee P_n \vee Q_2 \vee \ldots \vee Q_m$
- Examples
  - P and $\neg$ P $\vee$ Q : derive Q (Modus Ponens)
  - ($\neg$ P $\vee$ Q) and ($\neg$ Q $\vee$ R) : derive $\neg$ P $\vee$ R
  - P and $\neg$ P : derive False [contradiction!]
  - (P $\vee$ Q) and ($\neg$ P $\vee \neg$ Q) : derive True

```
function PL-RESOLUTION(KB, α) returns true or false
    clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
    new ← { }
    loop do
        for each C_i, C_j in clauses do
            resolvents ← PL-RESOLVE(C_i, C_j)
            if resolvents contains the empty clause then return true
            new ← new ∪ resolvents
        if new ⊆ clauses then return false
        clauses ← clauses ∪ new
```

FOL:

Whereas propositional logic assumes the world contains facts, first-order logic (like natural language) assumes the world contains

Objects: people, houses, numbers, colors, baseball games, war

Relations: red, round, prime, brother of, bigger than, part of, comes between,

Syntax of FOL: Basic elements

- Constants John, 2, DIT,
- Predicates Brother, >,
- Functions Sqrt, LeftLegOf,
- Variables x, y, a, b,
- Connectives ¬, ⇒, ∧, ∨, ⇔
- Equality =
- Quantifiers ∀, ∃

Atomic sentences:

Atomic sentence = *predicate* ($term_1$,...,$term_n$) or $term_1 = term_2$

Term = *function* ($term_1$,...,$term_n$) or *constant* or *variable*

Example:

Brother(John,RichardTheLionheart) > (Length(LeftLegOf(Richard)), Length(LeftLegOf(John)))

Complex sentences:

Complex sentences are made from atomic sentences using connectives

$\neg S$, $S_1 \wedge S_2$, $S_1 \vee S_2$, $S_1 \Rightarrow S_2$, $S_1 \Leftrightarrow S_2$,

Example:

Sibling(John,Richard) ⇒ *Sibling(Richard,John)*
>(1,2) ∨ ≤ (1,2)
>(1,2) ∧ ¬ >(1,2)

Universal quantification:

- ∀<*variables*> <*sentence*>

Everyone at SVCET is smart:

∀x At(x,SVCET) ⇒ Smart(x)

∀x *P* is true in a model *m* iff *P* is true with *x* being each possible object in the model

Existential quantification:

- ∃<*variables*> <*sentence*>

Someone at SVCET is smart:

∃*x* At(x,SVCET) ∧ Smart(x)$

∃*x* *P* is true in a model *m* iff *P* is true with *x* being some possible object in the model

Quantifier duality:

each can be expressed using the other

∀x Likes(x,IceCream) is same as ¬∃x ¬Likes(x,IceCream)

∃x Likes(x,Coffee) is same as ¬∀x ¬Likes(x,Coffee)

Equality:

- $term_1 = term_2$ is true under a given interpretation if and only if $term_1$ and $term_2$ refer to the same object

Example definition of *Sibling* in terms of *Parent*:

$\forall x,y\ Sibling(x,y) \Leftrightarrow [\neg(x = y) \wedge \exists m,f \neg (m = f) \wedge Parent(m,x) \wedge Parent(f,x) \wedge Parent(m,y) \wedge Parent(f,y)]$

The kinship domain Using FOL:

- Brothers are siblings

  $\forall x,y\ Brother(x,y) \Leftrightarrow Sibling(x,y)$

- One's mother is one's female parent

  $\forall m,c\ Mother(c) = m \Leftrightarrow (Female(m) \wedge Parent(m,c))$

- "Sibling" is symmetric

  $\forall x,y\ Sibling(x,y) \Leftrightarrow Sibling(y,x)$

Knowledge engineering in FOL

1. Identify the task
2. Assemble the relevant knowledge
3. Decide on a vocabulary of predicates, functions, and constants
4. Encode general knowledge about the domain
5. Encode a description of the specific problem instance
6. Pose queries to the inference procedure and get answers
7. Debug the knowledge base

HORN CLAUSE:

- A Horn clause is a sentence of the form:

  $(\forall x)\ P_1(x) \wedge P_2(x) \wedge \ldots \wedge P_n(x) \rightarrow Q(x)$

John likes to eat everything.

$\forall X\ food(X) \rightarrow likes(john,X)$

John likes at least one dish Jane likes.

$\exists F\ food(F) \wedge likes(jane, F) \wedge likes(john, F)$

Everybody likes some food.

$\forall X\ \exists F\ food(F) \wedge likes(X,F)$

There is a food that everyone likes.

$$\exists F \; \forall X \; food(F) \wedge likes(X,F)$$

Whenever someone eats a spicy dish, they're happy.

$$\forall X \; \exists F \; food(F) \wedge spicy(F) \wedge eats(X,F) \rightarrow happy(X)$$

John's meals are spicy.

$$\forall X \; meal\text{-}of(John,X) \rightarrow spicy(X)$$

Every city has a dogcatcher who has been bitten by every dog in town.

$$\forall T \; \exists C \; \forall D \; city(C) \rightarrow (dogcatcher(C,T) \wedge (dog(D) \wedge lives\text{-}in(D, T) \rightarrow bit(D, C)))$$

For every set x, there is a set y, such that the cardinality of y is greater than the cardinality of x.

$$\forall X \; \exists Y \; \exists U \; \exists V \; set(X) \rightarrow (set(Y) \wedge cardinality(X,U) \wedge cardinality(Y, V) \wedge greater\text{-}than(V,U))$$

Generalized Modus Ponens:

$$\forall x \; King(x) \wedge Greedy(x) \Rightarrow Evil(x)$$

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{Subst(\theta,q)}$$

Example:

p_1' is *King(John)*	p_1 is *King(x)*
p_2' is *Greedy(y)*	p_2 is *Greedy(x)*
θ is {x/John,y/John}	q is *Evil(x)*
Subst(θ,q) is *Evil(John)*	

- Implicit assumption that all variables universally quantified

- GMP is sound
  Only derives sentences that are logically entailed

- GMP is complete for a KB consisting of definite clauses
  Complete: derives all sentences that entailed

- Definite clause: disjunction of literals of which exactly 1 is positive,
  e.g., King(x) AND Greedy(x) -> Evil(x)
  NOT(King(x)) OR NOT(Greedy(x)) OR Evil(x)

- Forward-chaining
  - Uses GMP to add new atomic sentences
  - Useful for systems that make inferences as information streams in
  - Requires KB to be in form of first-order definite clauses
- Backward-chaining
  - Works backwards from a query to try to construct a proof
  - Can suffer from repeated states and incompleteness
  - Useful for query-driven inference
- Resolution-based inference (FOL)
  - Refutation-complete for general KB
    - Can be used to confirm or refute a sentence p (but not to generate all entailed sentences)
  - Requires FOL KB to be reduced to CNF
  - Uses generalized version of propositional inference rule

Unification:

- Unification is a **"pattern-matching"** procedure
  - Takes two atomic sentences, called literals, as input
  - Returns "Failure" if they do not match and a substitution list, θ, if they do
- That is, *unify(p,q) = θ* means *subst(θ, p) = subst(θ, q)* for two atomic sentences, *p* and *q*
- **θ** 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

Unification Algorithm:

```
procedure unify(p, q, θ)
        Scan p and q left-to-right and find the first corresponding  terms where p and q
"disagree" (i.e., p and q not equal)
If there is no disagreement, return θ  (success!)
   Let r and s be the terms in p and q, respectively,  where disagreement first occurs
        If variable(r) then {
                Let θ = union(θ, {r/s})
                Return unify(subst(θ, p), subst(θ, q), θ)
                        }
        else if variable(s) then {
                Let θ = union(θ, {s/r})
                Return unify(subst(θ, p), subst(θ, q), θ)
                        }
        else return "Failure"
end
```

Forward Chaining Algorithm

```
function FOL-FC-ASK(KB, α) returns a substitution or false
  repeat until new is empty
    new ← { }
    for each sentence r in KB do
      (p1 ∧ ... ∧ pn ⇒ q) ← STANDARDIZE-APART(r)
      for each θ such that (p1 ∧ ... ∧ pn)θ = (p'1 ∧ ... ∧ p'n)θ
                  for some p'1, ..., p'n in KB
        q' ← SUBST(θ, q)
        if q' is not a renaming of a sentence already in KB or new then do
          add q' to new
          φ ← UNIFY(q', α)
          if φ is not fail then return φ
    add new to KB
  return false
```

EXAMPLE :

- The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

it is a crime for an American to sell weapons to hostile nations:

$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$

Nono … has some missiles,
i.e., ∃x Owns(Nono,x) ∧ Missile(x):
$Owns(Nono,M_1)$ and $Missile(M_1)$

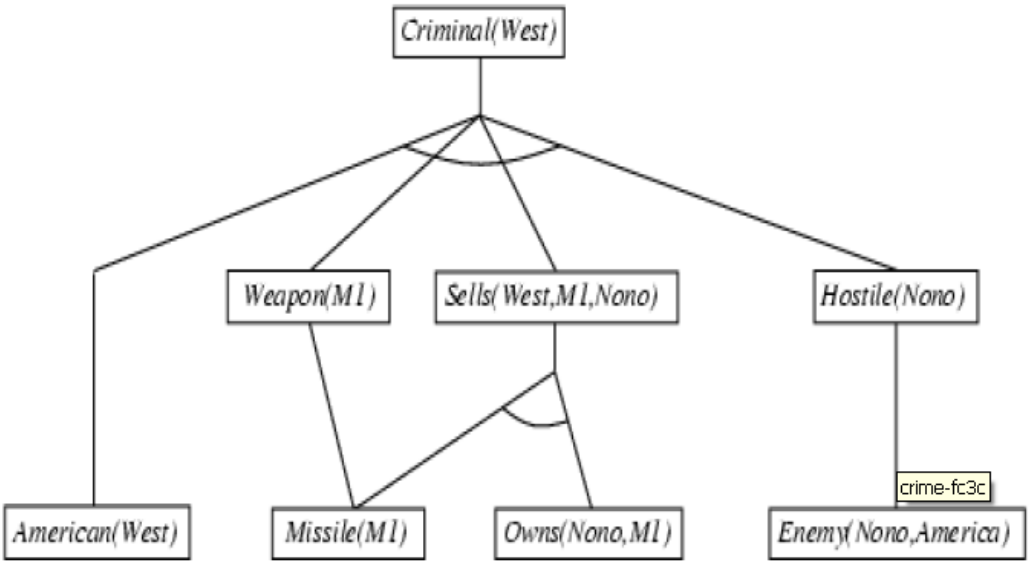
All of its missiles were sold to it by Colonel West
$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$

Missiles are weapons:
$Missile(x) \Rightarrow Weapon(x)$

An enemy of America counts as "hostile":
$Enemy(x,America) \Rightarrow Hostile(x)$

West, who is American …
$American(West)$

The country Nono, an enemy of America …
$Enemy(Nono,America)$



Backward chaining:

- Backward-chaining deduction using GMP is also complete for KBs containing only Horn clauses
- Proofs start with the goal query, find rules with that conclusion, and then prove each of the antecedents in the implication
- Keep going until you reach premises
- Avoid loops: check if new subgoal is already on the goal stack
- Avoid repeated work: check if new subgoal
  - Has already been proved true
  - Has already failed

it is a crime for an American to sell weapons to hostile nations:

$$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

Nono … has some missiles,

$$\exists x\ Owns(Nono,x) \wedge Missile(x)$$

$Owns(Nono,M_1)$ *and* $Missile(M_1)$

all of its missiles were sold to it by Colonel West

$$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$$

Missiles are weapons:

$$Missile(x) \Rightarrow Weapon(x)$$

An enemy of America counts as "hostile":

$$Enemy(x,America) \Rightarrow Hostile(x)$$

West, who is American

American(West)

The country Nono, an enemy of America …

Enemy(Nono,America)

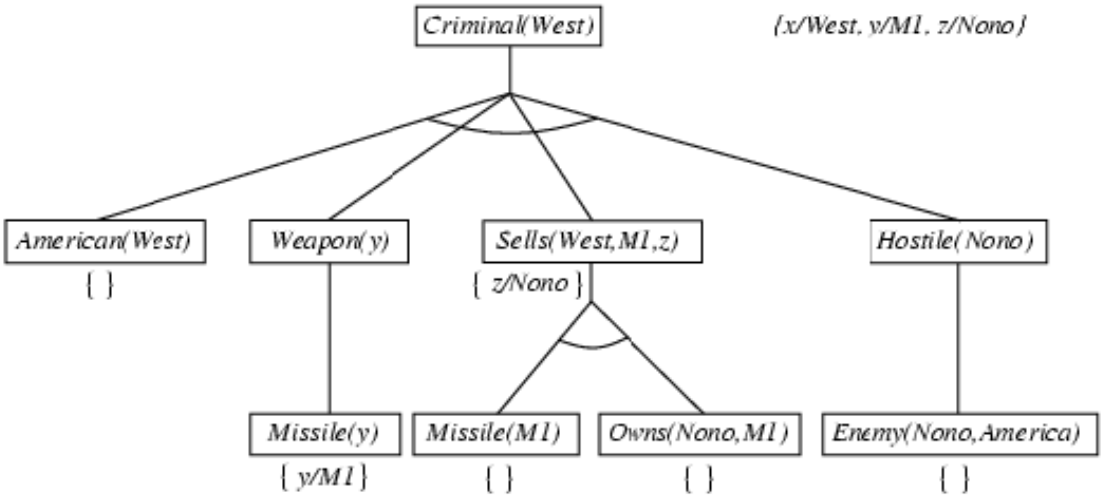
Can be converted to CNF

Query: Criminal(West)?

First Order Logic: (FOL)

function FOL-BC-ASK(KB, goals, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    goals, a list of conjuncts forming a query
    θ, the current substitution, initially the empty substitution { }
  local variables: *ans*, a set of substitutions, initially empty

  if *goals* is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST(*goals*))
  for each r **in** KB where STANDARDIZE-APART(r) = $(p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
    ans $\leftarrow$ FOL-BC-ASK(KB, $[p_1, \ldots, p_n | \text{REST}(goals)]$, COMPOSE($\theta$, θ')) $\cup$ *ans*
  return *ans*



Resolution in first-order logic

- Given sentences

$$P_1 \vee \ldots \vee P_n$$
$$Q_1 \vee \ldots \vee Q_m$$

- in *conjunctive normal form:*
  - each P_i and Q_i is a literal, i.e., a positive or negated predicate symbol with its terms,
- if P_j and $\neg Q_k$ unify with substitution list θ, then derive the resolvent sentence
- subst(θ, $P_1 \vee \ldots \vee P_{j-1} \vee P_{j+1} \ldots P_n \vee Q_1 \vee \ldots Q_{k-1} \vee Q_{k+1} \vee \ldots \vee Q_m$)

Example

- from clause $P(x, f(a)) \vee P(x, f(y)) \vee Q(y)$
- and clause $\neg P(z, f(a)) \vee \neg Q(z)$
- derive resolvent $P(z, f(y)) \vee Q(y) \vee \neg Q(z)$
- using $\theta = \{x/z\}$

Converting sentences to CNF

1. Eliminate all $\leftrightarrow$ connectives

$(P \leftrightarrow Q) \Rightarrow ((P \rightarrow Q) \wedge (Q \rightarrow P))$

2. Eliminate all $\rightarrow$ connectives

$(P \rightarrow Q) \Rightarrow (\neg P \vee Q)$

3. Reduce the scope of each negation symbol to a single predicate

$\neg \neg P \Rightarrow P$
$\neg(P \vee Q) \Rightarrow \neg P \wedge \neg Q$
$\neg(P \wedge Q) \Rightarrow \neg P \vee \neg Q$
$\neg(\forall x)P \Rightarrow (\exists x)\neg P$
$\neg(\exists x)P \Rightarrow (\forall x)\neg P$

4. Standardize variables: rename all variables so that each quantifier has its own unique variable name

Converting sentences to clausal form Skolem constants and functions

5. Eliminate existential quantification by introducing Skolem constants/functions

$(\exists x)P(x) \Rightarrow P(c)$

c is a Skolem constant (a brand-new constant symbol that is not used in any other sentence)

$(\forall x)(\exists y)P(x,y) \Rightarrow (\forall x)P(x, f(x))$

since $\exists$ is within the scope of a universally quantified variable, use a Skolem function f to construct a new value that depends on the universally quantified variable

f must be a brand-new function name not occurring in any other sentence in the KB.
E.g., $(\forall x)(\exists y)\text{loves}(x,y) \Rightarrow (\forall x)\text{loves}(x,f(x))$
In this case, f(x) specifies the person that x loves

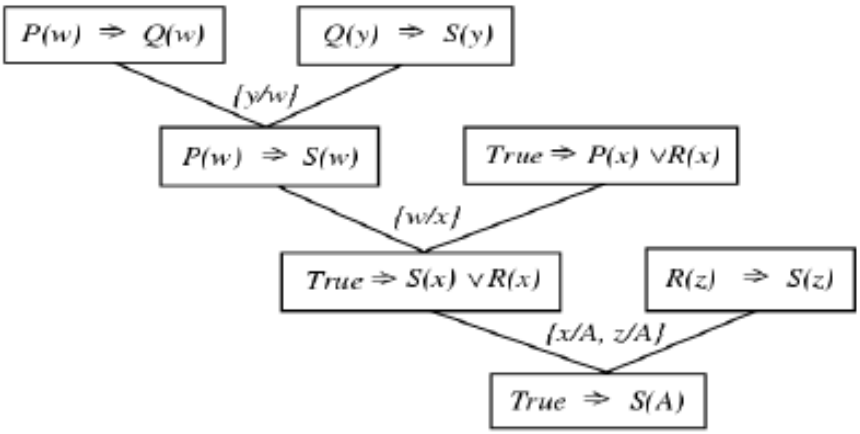
6. Remove universal quantifiers by (1) moving them all to the left end; (2) making the scope of each the entire sentence; and (3) dropping the "prefix" part
Ex: $(\forall x)P(x) \Rightarrow P(x)$

7. Put into conjunctive normal form (conjunction of disjunctions) using distributive and associative laws
$(P \wedge Q) \vee R \Rightarrow (P \vee R) \wedge (Q \vee R)$
$(P \vee Q) \vee R \Rightarrow (P \vee Q \vee R)$

8. Split conjuncts into separate clauses

9. Standardize variables so each clause contains only variable names that do not occur in any other clause

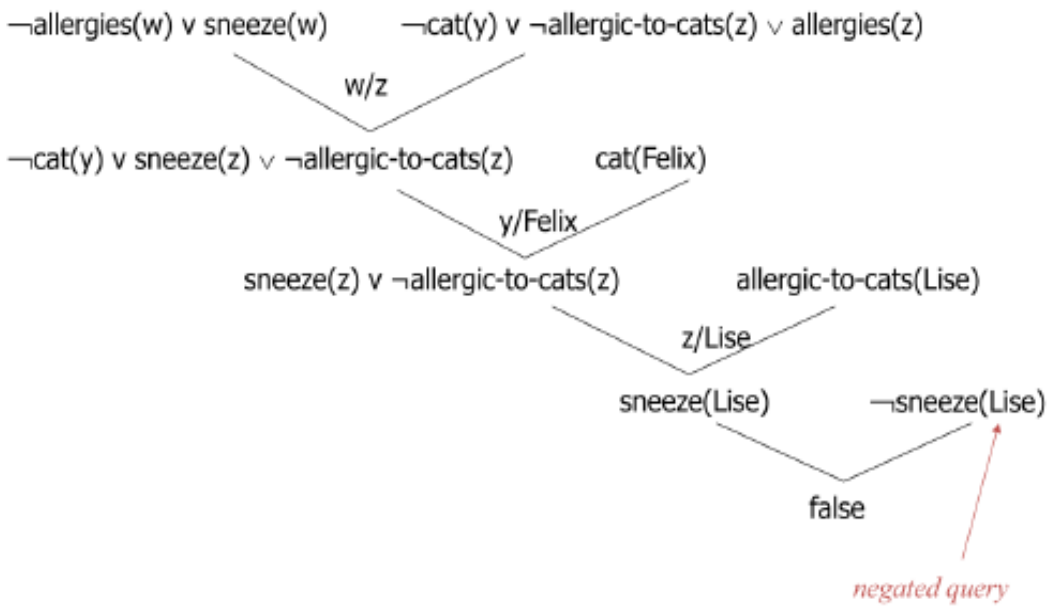
Resolution Proof



Resolution by refutation:

- Given a consistent set of axioms KB and goal sentence Q, show that KB |= Q
- Proof by contradiction: Add ¬Q to KB and try to prove false.
  i.e., (KB |- Q) ↔ (KB ∨ ¬Q |- False)
- Resolution is refutation complete: it can establish that a given sentence Q is entailed by KB, but can't (in general) be used to generate all logical consequences of a set of sentences
- Also, it cannot be used to prove that Q is not entailed by KB.
- Resolution won't always give an answer since entailment is only semidecidable
  And you can't just run two proofs in parallel, one trying to prove Q and the other trying to prove ¬Q, since KB might not entail either one

Refutation resolution proof tree



- **How to convert FOL sentences to conjunctive normal form (a.k.a. CNF, clause form): normalization and skolemization**
- **How to unify two argument lists, i.e., how to find their most general unifier (mgu) q: unification**
- **How to determine which two clauses in KB should be resolved next (among all resolvable pairs of clauses) : resolution (search) strategy**

it is a crime for an American to sell weapons to hostile nations:

$$American(x) \wedge Weapon(y) \wedge Sells(x,y,z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

Nono … has some missiles,

$$\exists x\ Owns(Nono,x) \wedge Missile(x)$$

$Owns(Nono,M_1)$ and $Missile(M_1)$

all of its missiles were sold to it by Colonel West

$$Missile(x) \wedge Owns(Nono,x) \Rightarrow Sells(West,x,Nono)$$

Missiles are weapons:

$$Missile(x) \Rightarrow Weapon(x)$$

An enemy of America counts as "hostile":

$$Enemy(x,America) \Rightarrow Hostile(x)$$

West, who is American

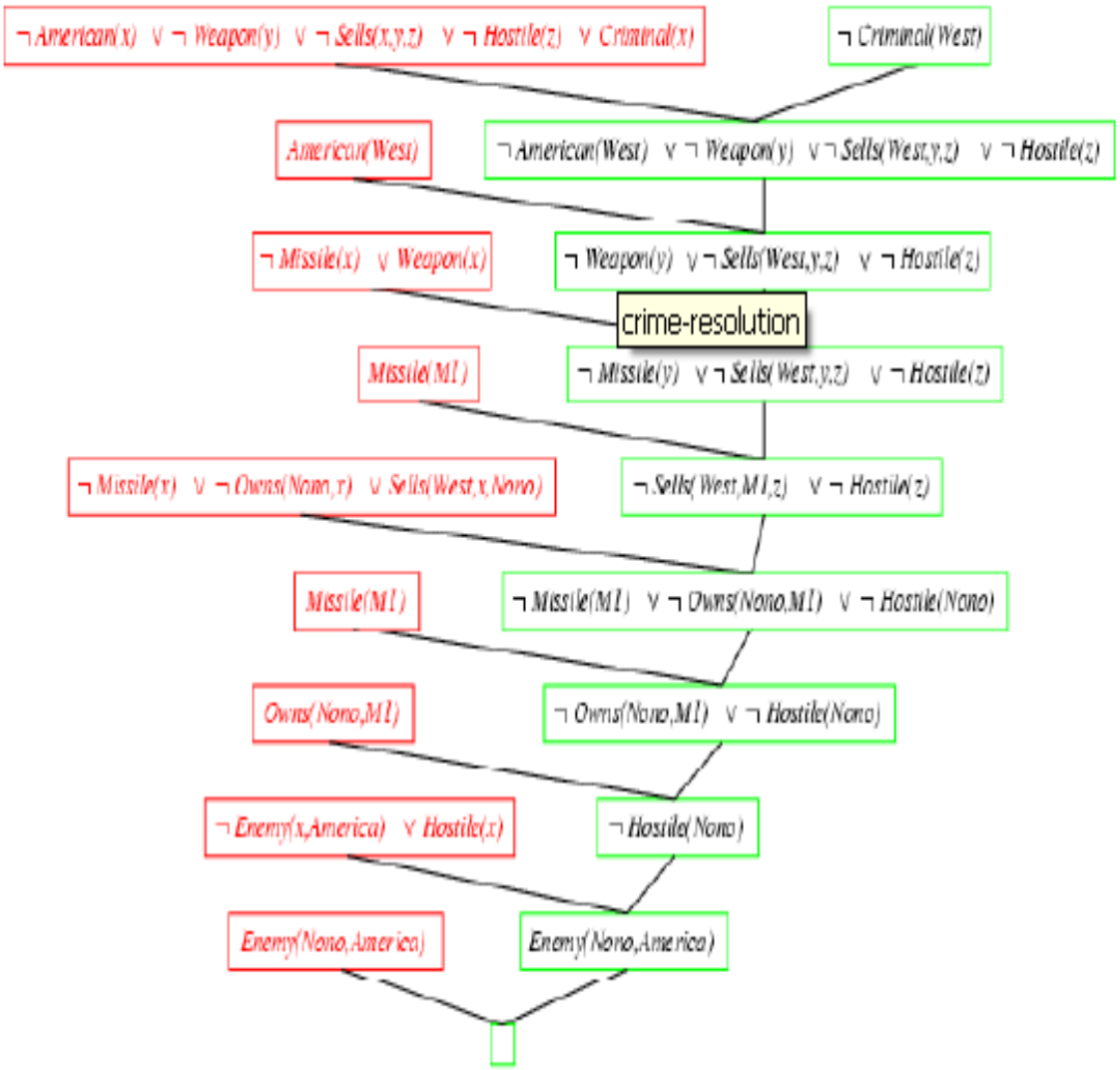
$$American(West)$$

The country Nono, an enemy of America …

$$Enemy(Nono,America)$$

Can be converted to CNF

Query: Criminal(West)?



STRUCTURED REPRESNTATION OF KNOWLEDGE

Representing knowledge using logical formalism, like predicate logic, has several advantages. They can be combined with powerful inference mechanisms like resolution, which makes reasoning with facts easy. But using logical formalism complex structures of the world, objects and their relationships, events, sequences of events etc. can not be described easily.

A good system for the representation of structured knowledge in a particular domain should posses the following four properties:

(i) Representational Adequacy:- The ability to represent all kinds of knowledge that are needed in that domain.

(ii) Inferential Adequacy :- The ability to manipulate the represented structure and infer new structures.

(iii) Inferential Efficiency:- The ability to incorporate additional information into the knowledge structure that will aid the inference mechanisms.

(iv) Acquisitional Efficiency :- The ability to acquire new information easily, either by direct insertion or by program control.

The techniques that have been developed in AI systems to accomplish these objectives fall under two categories:

1. Declarative Methods:- In these knowledge is represented as static collection of facts which are manipulated by general procedures. Here the facts need to be stored only one and they can be used in any number of ways. Facts can be easily added to declarative systems without changing the general procedures.

2. Procedural Method:- In these knowledge is represented as procedures. Default reasoning and probabilistic reasoning are examples of procedural methods. In these, heuristic knowledge of "How to do things efficiently "can be easily represented.

In practice most of the knowledge representation employ a combination of both. Most of the knowledge representation structures have been developed to handle programs that handle natural

language input. One of the reasons that knowledge structures are so important is that they provide a way to represent information about commonly occurring patterns of things . such descriptions are some times called schema. One definition of schema is

“Schema refers to an active organization of the past reactions, or of past experience, which must always be supposed to be operating in any well adapted organic response”.

By using schemas, people as well as programs can exploit the fact that the real world is not random. There are several types of schemas that have proved useful in AI programs. They include

(i) Frames:- Used to describe a collection of attributes that a given object
a. possesses (eg: description of a chair).

(ii) Scripts:- Used to describe common sequence of events
(eg:- a restaurant scene).

(iii) Stereotypes :- Used to described characteristics of people.

(iv) Rule models:- Used to describe common features shared among a set of rules in a production system.

Frames and scripts are used very extensively in a variety of AI programs. Before selecting any specific knowledge representation structure, the following issues have to be considered.

(i) The basis properties of objects , if any, which are common to every problem domain must be identified and handled appropriately.

(ii) The entire knowledge should be represented as a good set of primitives.

(iii) Mechanisms must be devised to access relevant parts in a large knowledge base.

Planning

The task of coming up with a sequence of action to achieve a goal is called Planning

It consists of

States - Conjunction of Literals

Action - It depends on

- Pre Condition
- Effects

Two Types of Planning

Classical Planning

Non classical Planning

Classical Planning : The environment which are fully observable, deterministic, finite, static & discrete

Non classical Planning : The environment which is Partially observable or stochastic environment

No.

The straight forward approach is state space search. Description of action in Planning Problem specify both preconditions & effects but in state space it is possible to search in either forward from the initial state or backward from the Goal. Explicit goal & action representation to drive heuristic automatically

Forward & Backward Search

Forward Search

Planning with Forward State space search called Progression Planning

. The initial state of the Search is the initial state from the Planning Problem

No. 3.3

In general, each state will be a set of Positive ground literals. literals which are not appearing are False.

(ii) The action that are applicable to a state are all those whose preconditions are satisfied

(iii) The Goal test checks whether the state satisfies the goal of the Planning Problem

(iv) The step cost of each action is typically 1. Although it would be easy to allow different costs for different action

* Forward Search does not address the irrelevant action problem

No. 3.4.

Algorithm

Forward Search

Progress (State, goal, Action, Path)

if state satisfies goal

return Path;

else a = choose (Action)

Preconditions (a) satisfied in state

if no such a, then return false

else return

Progress (apply (a, state), goals,

actions Concatenate (Path, a)

Backward Search

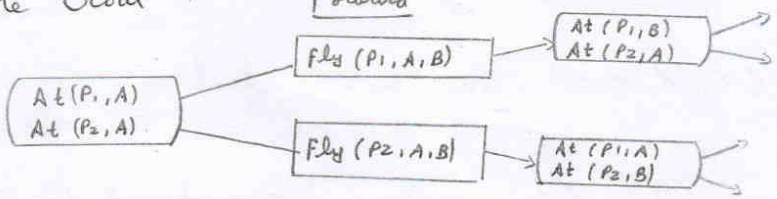
Backward Search can be difficult to implement when the Goal States are described by a set of constraints

No. 35

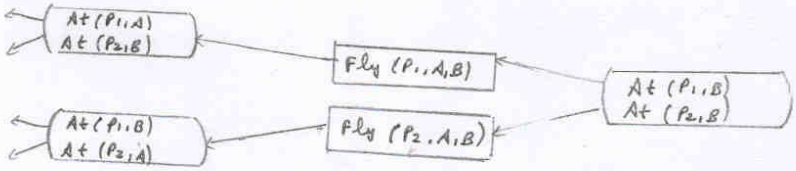
rather than being listed Explicitly.

The advantage of backward search is to consider only relevant action. An action is relevant to a Conjunctive goal if it is achieves one of the conjunction of the Goal

Forward



Backward



Algorithm for Backward.

Regress (IS, Current Goal, Actions, Path)

if IS satisfies current Goals, then

return Path

No. 36

else a = choose (action) ST Some effects of a satisfies one of Current Goal

if no such a, then return failure

if some effect of a Contradicts some current Goal;

then return failure

CG' = CG - efforts [a] + Pre Condition [a]

of CG < CG' then return failure

Reg ws (IS, CG', action, concatenate (a, Path))

16/3/15'

3.7

GRAPHPLAN :

The graphplan algorithm repeatedly adds a level to a planning graph with EXPAND-GRAPH. Once all the goals show up as non-mutex in the graph, Graphplan calls Extract-Solution to search for a plan that solves the problem. If that fails, it expands another level and tries again, terminating with failure when there is no reason to go on.

Algorithm :

```
function GRAPHPLAN(problem) returns solution or failure
    graph ← Initial-Planning-Graph(problem)
    goals ← Conjuncts(problem.Goal)
    for tl = 0 to ∞ do
        if goals all non-mutex in St of graph then
            solution ← Extract-Solution(graph, goals, NumLevels(graph), nogoods)
            if solution ≠ failure then return solution
        if graph and nogoods have both leveled off then return failure
        graph ← Expand-Graph(graph, problem)
```

Example :

Init(Tire(Flat) ∧ Tire(Spare) ∧ At(Flat, Axle) ∧ At(Spare, Trunk))
Goal(At(Spare, Axle))
Action(Remove(obj, loc),
    PRECOND: At(obj, loc)
    EFFECT: ¬At(obj, loc) ∧ At(obj, Ground))
Action(PutOn(t, Axle),
    PRECOND: Tire(t) ∧ At(t, Ground) ∧ ¬At(Flat, Axle)
    EFFECT: ¬At(t, Ground) ∧ At(t, Axle))
Action(LeaveOvernight,
    PRECOND:
    EFFECT: ¬At(Spare, Ground) ∧ ¬At(Spare, Axle) ∧
        ¬At(Spare, Trunk) ∧ ¬At(Flat, Ground) ∧
        ¬At(Flat, Axle) ∧ ¬At(Flat, Trunk))

10

In the above problem we have the full component of actions, it is worthwhile to look at some of the examples of mutex relations and their causes:

(i) **Inconsistent Effects:**

Remove (Spare, Trunk) is mutex with Leave Overnight because one has effect At (Spare, Ground) and other has its negation.

(ii) **Interference:**

Remove (Flat, Axle) is mutex with Leave Overnight because one has the precondition At (Flat, Axle) and other

(iii) **Competing Needs:**

PutOn (Spare, Axle) is mutex with Remove (Flat, Axle) because one has At (Flat, Axle) as a precondition and other has its negation.

(iv) **Inconsistent Support:**

At (Spare, Axle) is mutex with At (Flat, Axle in S_2 because only way of achieving At (Spare, Axle) is by PutOn (Spare, Axle) and that is mutex with the persistence action that is the only way of achieving At (Flat, Axle). Thus mutex relations detect immediate conflict that arises from trying to put two objects in same place at same time.

Extract Solution:

We can formulate extract solution as a Boolean constraint satisfication problem (CSP) where variables are actions at each levels, the values are in or out of the plan and the constraints are the mutexes and the need to satisfy each goal and precondition.

Alternatively we can define Extract-Solution as a backward search problem, where each state in search contains a pointer to a level in planning graph and a set of unsatisfied goals.

Termination of Graphplan:

Graphplan will infact terminate and **return failure when there is no solution**. We can terminate with failure because there is no possibility of a subsequent change that could add a solution.

The properties are:

(i) **Literals increase monotonically:**

Once a literals appears at a given level, it will appear at all subsequent levels. This is because of the persistence actions; once a literals shows up, persistence action cause it to stay forever.

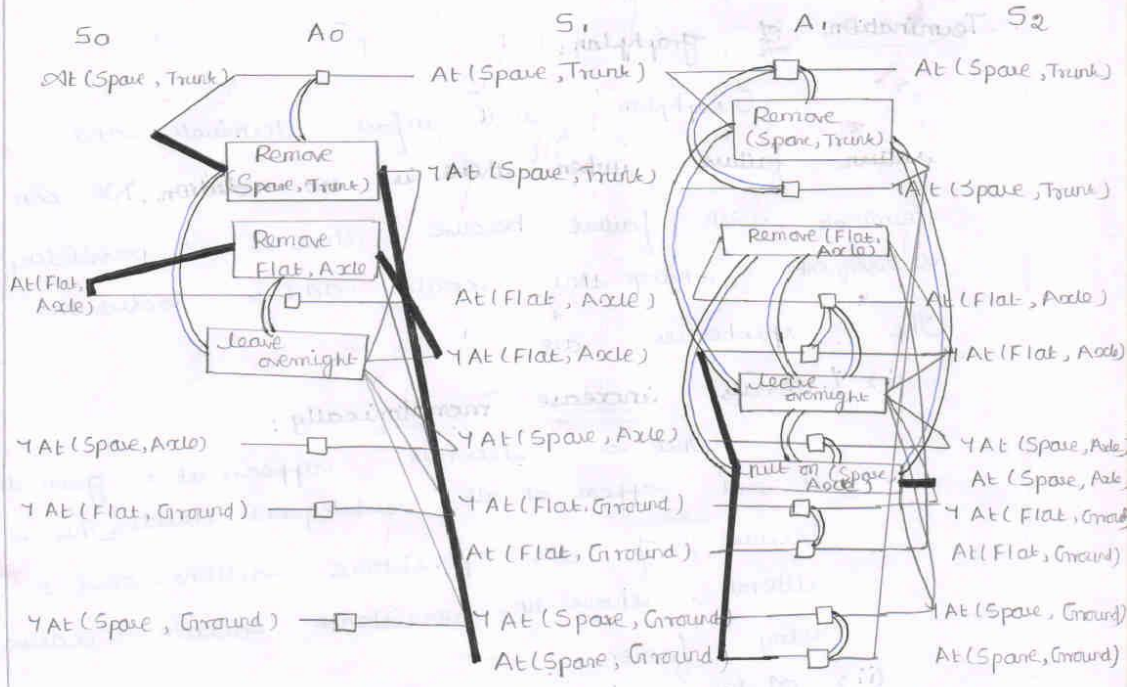
(ii) **Actions increase monotonically:**

Once an action appears at a given level, it will appear at all subsequent level. This is a consequence of monotonic increase of literals.

(iii) **Mutexes decrease monotonically:**

If two actions are mutex at given level A_i, then they will also be mutex for all previous levels at which they both appear.

The following diagram describes about the planning graph for spare tire problem after expansion to level S_2.



- Mutex lines are shown as double lines.
- Solution is indicated by Bold lines & outlines.

Planning graph for heuristic estimation:

A planning graph, once constructed is a rich source of information about problem. First if any goal literal fails to appear in final level of graph then problem in unsolvable.

Second we can estimate cost of achieving any goal literal g_i from state s as the level at which g_i first appears in planning graph constructed from initial state s. We call this as level cost of g_i.

Approaches:

3.11

The following are the three approaches to estimate the cost of a conjuction of goals.

- Max level
- level sum
- set-level

(i) Max level:

The max-level heuristic simply takes maximum level cost of any of goals, this is admissible but not necessarily accurate.

(ii) Level Sum:

In level sum heuristic, the following subgoals are to be independence assumption, returns the sum of level cost of goals, this can be inadmissible but works well in practice for problems that are largely decomposable.

(iii) Set level:

The set-level heuristic finds the levels at which all literals in the conjuctive goal appear in the planning graph without any pair of them being mutually exclusive.

Hierarchical Task Networks (or) HTN Planning :-

⇒ In Hierarchical Task Networks (or) HTN planning, it has some primitive actions. The following are the primitive actions in HTN planning

* Full Observability and Determinism
* Availability of set of actions
* Standard precondition – effect Schemas.

⇒ The Hierarchical Decomposition is implemented in this planning, is to reduce the complexity.

⇒ At each level of hierarchy, a computational task is reduced to a small number of activities at the next lower level.

⇒ The computational task of arranging these activities is low.

⇒ Hierarchical Task Network (HTN) planning uses a refinement of actions through decomposition

Types of HTN Planning :-

There are two types of HTN planning. They are,

* Pure HTN Planning
* Hybrid HTN Planning.

Hierarchical Planning :-

⇒ Hierarchical Planning is a plan at higher level of abstraction

⇒ Here, the planning can occur both before and during the execution of plan.

⇒ In Hierarchical Planning, particular action will remain at an abstract level prior to the execution phase.

⇒ In Hierarchical Planning, we concentrate on the aspect of Hierarchical Decomposition

Hierarchical Decomposition :-

⇒ Hierarchical Decomposition is an idea that pervades almost all attempts to manage complexity.

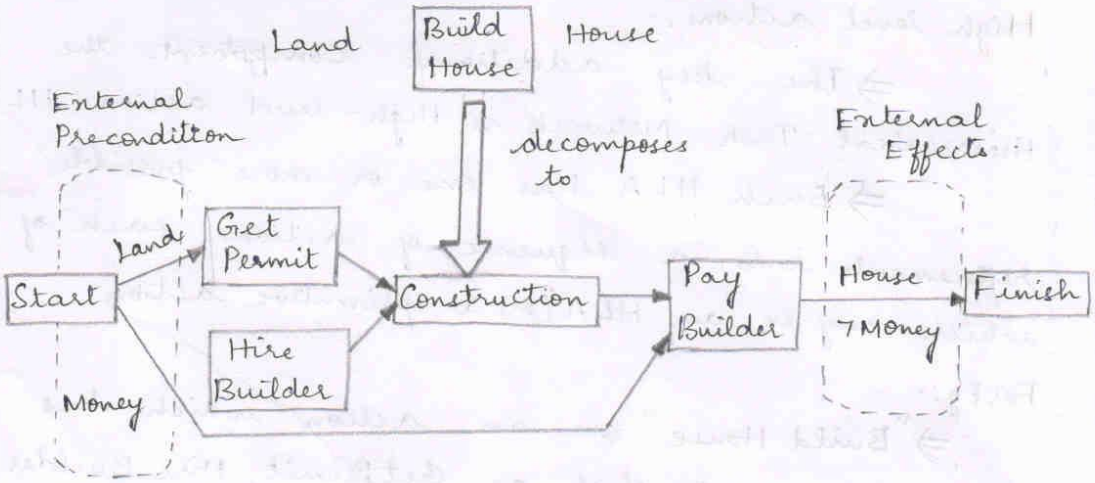
Eg:- armies operate on hierarchy of units, complex software is created from a hierarchy of subroutines or object classes.

⇒ The basic formalism we adopt to understand hierarchical Decomposition comes from the area of Hierarchical Task Networks (HTN) planning.

Start action supplies all preconditions of actions, not supplied by other actions } = External Preconditions

Finish action has all effects of actions not present in other actions } = External Effects

Example:- "Building a House"



Action (Buyland, PRECOND: Money, EFFECT: Land ∧ ¬ Money)

Action (GetLoan, PRECOND: Goodcredit, EFFECT: Money ∧ Mortgage)

Action (BuildHouse, PRECOND: Land, EFFECT: House)

Action (GetPermit, PRECOND: Land, EFFECT: Permit)

Action (HireBuilder, PRECOND: Contract)

Action (Construction, PRECOND: Permit ∧ Contract, EFFECT: House Built ∧ ¬ Permit

Action (PayBuilder, PRECOND : Money ∧ HouseBuilt,

EFFECT : ¬Money ∧ House ∧ ¬(Contract)

Decompose (BuildHouse, Plan ::

STEPS { S1 : GetPermit, S2 : Hire Builder, S3 : Construction,
S4 : PayBuilder }

ORDERINGS : { Start < S1 < S3 < S4 < Finish, Start < S2 < S3 }

LINKS : { Start $\xrightarrow{Land}$ S1, Start $\xrightarrow{Money}$ S4,
S1 $\xrightarrow{Permit}$ S3, S2 $\xrightarrow{Contract}$ S3, S3 $\xrightarrow{HouseBuilt}$ S4
S4 $\xrightarrow{house}$ Finish, S4 $\xrightarrow{\neg Money}$ Finish }

High level actions :-

⇒ The key additional concept in the Hierarchical Task Network is High-level action (HLA).

⇒ Each HLA has one or more possible refinements into a sequence of actions, each of which may be an HLA (or) a primitive action

For Eg :-

"Build House" is an action, which has many refinements such as GetPermit, Hire Builder, Construction, Pay Builder in the example mentioned above.

⇒ High level Plan achieves the goal from a given state, if atleast one of its implementations achieves the goal from that state

Partially Ordered Planning :-

⇒ Partially ordered plan is a set of actions and a set of constraints of the form Before (a_i, a_j), saying that one action occurs before another.

⇒ Partially ordered plans are created by a search through the space of plans rather than through the state space.

⇒ It starts with the empty plan consisting of just the initial state and Goal.

⇒ It has no actions in between.

Planning to be possible in four components :-

⇒ The partially Ordered planning could be possible by using the following four components.

1. Action :

* It makes up the step, by using the two dummy actions Start, Stop.

2. Ordering Constraint :-

* The Ordering Constraint in the Partially Ordered Planning is represented as below.

$A \prec B$ (A before B)

* The constraint states that, A must be executed before B. But, not neccessarily immediately

3. Casual Link :-

* Casual Link in Partially Ordered Planning is represented as below.

$A \xrightarrow{P} B$ (A achieves P for B)

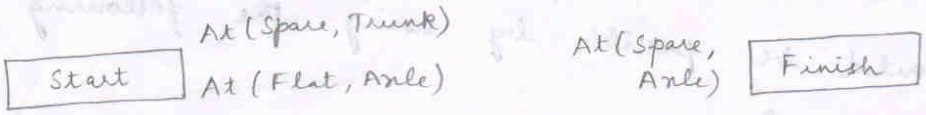
4. Open precondition :-

* A set of preconditions are open, if they are not achieved by some actions in the plan.

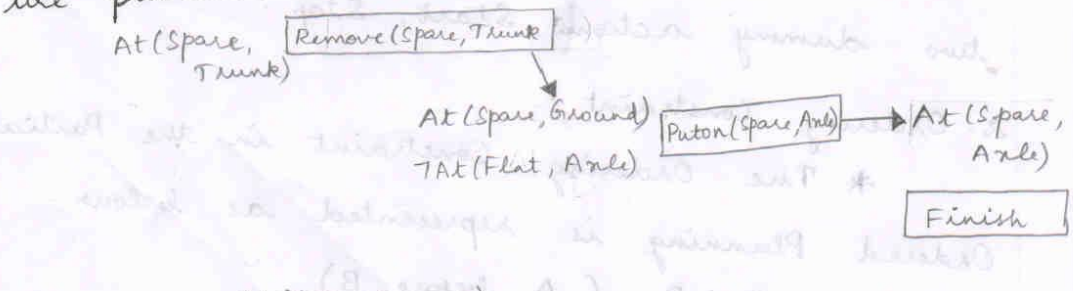
Example :-

Let us consider the spare tire problem for an example,

(i) The tire problem expressed as an empty plan:

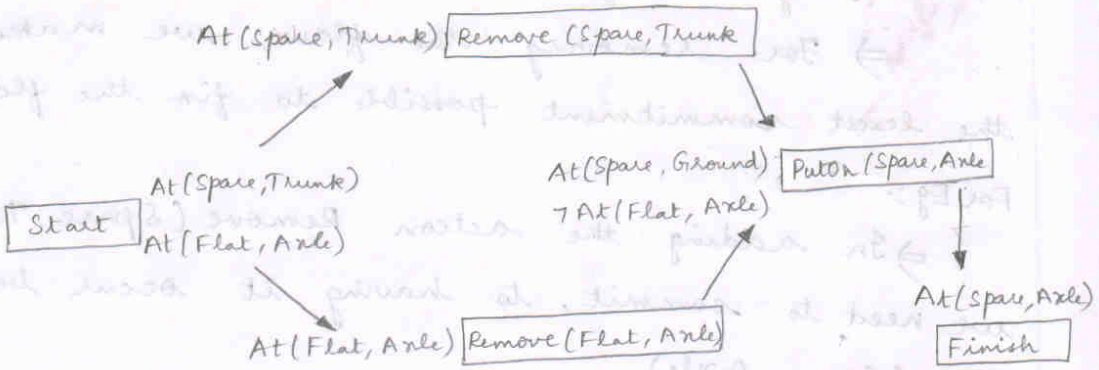


(ii) An incomplete partially ordered plan for the tire problem :



(Boxes represent Actions and arrows indicate that one action must occur before another)

(iii) A complete partially - ordered solution :- 3.18



Finding Solution for a problem using Partially Ordered Planning :-

1. Detecting Flaw :-

⇒ The Search procedure looks for a flaw in the plan, and makes an addition to the plan to correct the flaw.

⇒ A flaw is anything that keeps the partial plan from being a solution.

For Eg :-

⇒ One flaw present in the empty plane (i) is, no action achieves At (Spare, Axle)

⇒ One Way to correct the flaw is to insert into the plan, the action PutOn (Spare, Axle).

⇒ It also introduces some new flaws, "The preconditions of the new action are not achieved"

⇒ The Search keeps adding to the plan until all flaws are resolved. 3.19

(ii) Removing the flaw :-

⇒ For removing the flaw, we make the least commitment possible to fix the flaw.

For Eg :-

⇒ In adding the action Remove (Spare, Trunk) we need to commit, to having it occur before PutOn (Spare, Axle)

⇒ But, we make no other commitment that places it before or after other action.

(iii) Backtracking (if neccessary) :-

⇒ If again, there exists any flaw, move to the Step(i) (Detecting Flaw)

⇒ If no flaw exists, then return the solution of Partially Ordered Planning.

16/3/15

Artificial Intelligence. 3.20

Planning and acting in non-deterministic domains :-

* In an Uncertain environment, agent must use its percepts to discover what is happening while the plan is being executed.
* Agents have to deal with both incomplete and incorrect informations.
* Incompleteness arises because the world is partially observable, non-deterministics.
* Incorrectness arises because the world does not necessarily match my model of the world.
* The possibility of having complete or correct knowledge represents / depends on how much indeterminancy is there in the world.
* Bounded Indeterminancy - Actions can have unpredictable effects.
* Unbounded Indeterminacy - The set of possible preconditions or effects either is unknown or too large to be enumerated completely.

Four planning methods for handling indeterminacy :-

1. Sensorless Planning. } Bounded Indeterminacy
2. Conditional Planning }
3. Execution monitoring & replanning } Unbounded Indeterminacy
4. Continuous Planning }

Sensorless Planning :-

* Also called as Conformant Planning.
* It constructs standard, sequential plans that are to be executed without perceptions.
* This planning algorithm must ensure that the plan achieves the goal in all possible circumstances.
* This relies on coercion.

* Coercion :-

The idea that the world can be forced into a given state even when the agent has only partial information about the current state. 3.21

* Sensorless planning is often inapplicable due to coercion.

Conditional Planning :-

* Also called as contigency planning.
* It constructs a conditional plan with different branches for the different contigencies that could arise.
* The agents finds out which part of the plan to execute by including sensing actions in the plan to test for the appropriate conditions.
* The agent plans first and executes the plan that was produced.

Execution Monitoring and Replanning :-

* In this planning, the agent can use any of the planning preceding techniques to construct a plan.
* It uses execution monitoring to judge whether the plan has a provision for actual current situations or to be revised again.
* Replanning occurs when something goes wrong.

Continuous Planning :-

* Continuous planning is designed to persist over a lifetime.
* It can handle unexpected circumstances in the environment, even if these occurs in the middle of the constructing a plan.
* It also handle abandonment of goals and the creation of additional goals by goal formulations.

Critical path Method (CPM) :-

↳ To minimize a makespan, find the earliest start time for all the action consistent with the ordering constraint. [ordering constraints as a directed graph related to their action].

CPM :- CPM is a graph to determine the possible start and end times of each action.

↳ A path through a graph repersenting a partial - ordered plan in a linear ordered, action beigning with 'start' and ending with 'Finish'.

↳ In partial ordered structure : a plan is a set of action and set of constraints of a form Before (a_i, a_j). It cause some ambiguity, to overcome an ambiguity and minimize the makespan by using Critical path Method.

Critical Path :- Critical path is a path, whose total duration is longest, the path is said to be Critical because it determines the duration of the entire plan.

↳ Delaying the start of any action on the Critical plan slow down the whole plan. Critical path have a window size of time where they executed.

Slack :- The window is Specified in terms of an earliest possible start time (ES) and latest possible start time (LS). The quantity LS-ES is known as slack of an action. ES and LS times for action on the Critical path constitute a schedule for the problem.

The definition for ES and LS and also as the outline of dynamic-programming algorithm to compute them. 3.23

$ES(Start) = 0$

$ES(B) = \max_{A \prec B} ES(A) + Duration(A)$

$LS(Finish) = ES(Finish)$

$LS(A) = \min_{B \succ A} LS(B) - Duration(A)$.

Example :-

Jobs({AddEngine1 ≺ AddWheels1 ≺ Inspect1},
{AddEngine2 ≺ AddWheels2 ≺ Inspect2})

Resources (EngineHoists (1), WheelStation (1), Inspectors (2), LugNuts (500))

Action (AddEngine1, DURATION: 30,
USE : EngineHoists (1))

Action (AddEngine2, DURATION: 60,
USE : EngineHoists (1))

Action (AddWheels1, Duration: 15,
USE : WheelStation (1))

Action (AddWheels2, DURATION: 15,
USE : WheelStations (1))

Action (Inspecti, Duration : 10,
USE : Inspectors (1))

* The complexity of critical path algorithm is just $O(Nb)$ where N is the number of actions and b is the maximum branching factor into or out of an action.

1. Bayesian Network and Explain Inference by Enumeration.

Baye's Rule.

The Product rule can be written in two forms

$$P(a \wedge b) = P(a|b)\,P(b)$$
$$P(a \wedge b) = P(b|a)\,P(a)$$

Equating two right hands & divided by $P(a)$

$$\Rightarrow \boxed{P(b|a) = \frac{P(a|b)P(b)}{P(a)}}$$

This equation is known as Baye's rule (or) Baye's Theorem.

Bayesian Network: (used to solve large no. of variable)

A Data structure called bayesian Network to represent the dependencies among variable. It can represent essentially any full joint probability distribution & in many cases can do very concisely.

It is directed graph in which each node is annoted with quantitative probability information.

The Full specifications are

*) Each node corresponds to a random variable, which may be discrete (or) continous.

*) A set of directed links (or) arrows connects pair of nodes. If there is an arrow from node x to node Y, x is said to be parent of Y. The graph has no directed cycles. (DAG)

*) Each node x_i, has a conditional probability distribution $P(x_i | \text{Parents}(x_i))$ that quantifies the effect of the parents on the node.

Inference by enumeration.

It is one of the exact inference in Bayesian Network.

Any conditional probability can be computed by summing terms from the full joint distribution.

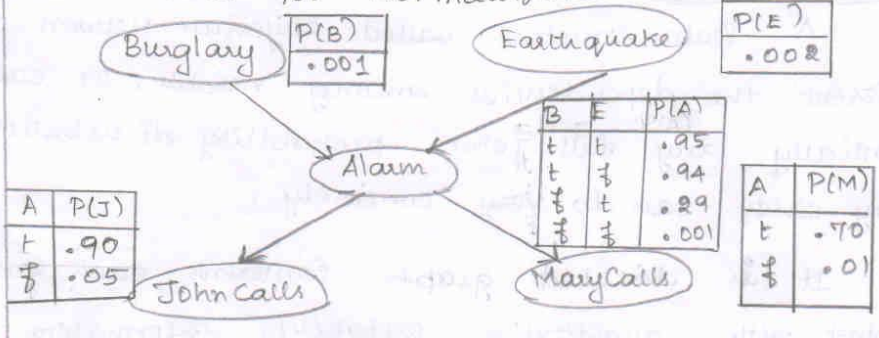
$$P(x/e) = \alpha P(x,e) = \alpha \sum_{y} P(x,e,y)$$

X denotes the query variable.

E denotes the set of evidence variable.

y denotes the hidden variable.

α is for Normalization.



A query can be answered using a Bayesian network by computing sums of products of conditional probabilities from the network.

Consider the query P(Burglary | JohnCalls = true, MaryCalls = true).

The Hidden variables in this query are Earthquake & Alarm.

Using Initial letters to shorten the expression.

$$\Rightarrow P(B|j,m) = \alpha P(B,j,m) = \alpha \sum_e \sum_a P(B,j,m,e,a)$$

Gives us an expression interms of CPT entries.

⇒ we do this just for Burglary = true;

$$P(b|j,m) = \alpha \sum \sum p(b)\, P(e)\, P(a|b,e)\, P(j|a)\, P(m|a)$$

To compute this expression, we have to add four terms, each computed my Multiplying 5 no.'s.

Complexity of the algorithm for a network with 'n' Boolean Variables is $O(n2^n)$

From Observation P(b) term is a constant so we move that Outside the summations over a and e. P(e) moved outside the Summation over a.

$$P(b|j,m) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e)\, P(j|a)\, P(m|a)$$

To evaluate this by looping & Multiplying CPT entries.

The Structure will be

Algorithm.

function ENUMERATION-ASK(X, e, bn) returns a distribution over X

Inputs: X, the query variable

e, observed values for variables E

bn, a Bayes net with variables $\{X\} \cup E \cup Y$

/* Y hidden Variable */

Q(X) ← a distribution over X, initially empty

for each value x_i of X do

$Q(x_i) \leftarrow$ ENUMERATE-ALL(bn, VARS, e_{x_i})

where e_{x_i} is e extended with $X = x_i$)

return NORMALIZE(Q(X))

function ENUMERATE-ALL(vars, e) returns a real number

if EMPTY?(vars) then return 1.0

$Y \leftarrow$ FIRST(vars)

if Y has value y in e

then return $P(y \mid parents(Y)) \times$ ENUMERATE-ALL(REST(vars), e)

else return $\sum_y P(y \mid parents(Y)) \times$ ENUMERATE-ALL(REST(vars), e_y)

where e_y is e extended with $Y = y$.

1) Bayesian Network:

A Bayesian network is a directed graph in which each node is annotated with quantitative probability information. The full specification is as follows:

* Each node corresponds to a random variables, which may be discrete or continous.
* A set of directed links (or) arrows connects pair of nodes.
* Each node x_i, has a conditional probability distribution.

Full joint distribution:

~~viewed as a piece of "syntax",~~ a

A Bayesian network is a directed acyclic graph with some numeric parameters attached to each node.

A generic entry in the joint distribution is the probability of a conjunction of particular assignments to each variable, such as

$P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n)$.

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} \Theta(x_i \mid \text{parents}(x_i))$$

where parents (x_i), denotes the values of Parents (x_i) that appear in $x_1, \ldots, x_n$. Thus each entry in the joint distribution is represented by the product of the appropriate elements of the conditional probability table(CPT) in the Bayesian Network.

⊛

$$P(j, m, a, \neg b, \neg e) = P(j|a)\, P(m|a)\, P(a \mid \neg b \wedge \neg e)\, P(\neg b)\, P(\neg e)$$

$$= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998$$

$$= 0.000628.$$

The full joint distribution can be used to answer any query about the domain.

A method for constructing Bayesian networks:

$$P(x_1, \ldots, x_n) = P(x_n \mid x_{n-1} \ldots x_1)\, P(x_{n-1}, \ldots x_1)$$

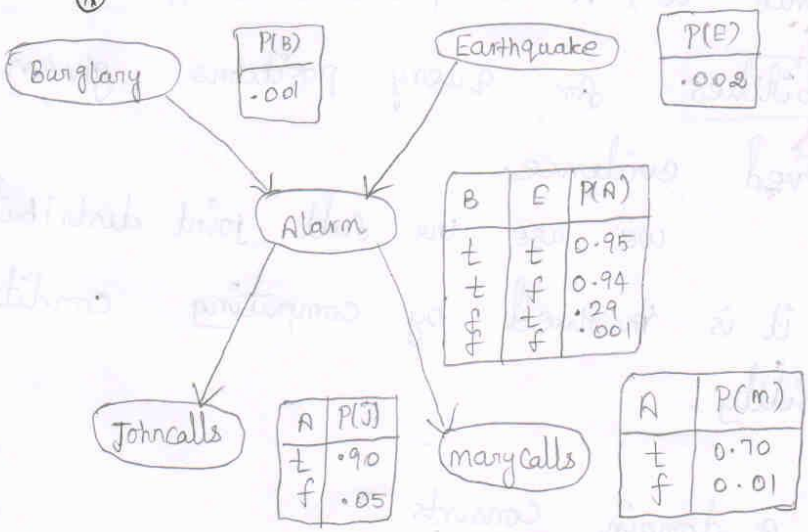
Then we repeat the process, reducing each conjunctive probability to a conditional probability and a smaller conjunction. we end up with big product:

$$P(x_1, \ldots x_n) = P(x_n \mid x_{n-1}, \ldots x_1)\, P(x_{n-1} \mid x_{n-2}, \ldots x_1)$$
$$\ldots P(x_2|x_1)\, P(x_1)$$

$$= \prod_{i=1}^{n} P(x_i \mid x_{i-1}, \ldots x_1).$$

This identity is called the chain rule.

(*)



The conditional distributions can be used for discrete variables, other representations including those suitable for continous variables, Conditional probability table to representing :-

Bayesian network is a correct representation of the domain only if each node is conditionally independent of its other predecessors in the ordering:

* Nodes
* Links

Inference using Full joint Distribution:

We introduce a method for probabilistic Inference ie) A computation of posterior probabilities for query positions given observed evidence.

We use the full joint distribution and it is invoked by computing conditional Probability.

eg:

A domain consists

* Toothache
* Cavity
* Catch

	tooth ache		¬toothache	
	Catch	¬catch	Catch	¬catch
Cavity	0.108	0.012	0.072	0.008
¬Cavity	0.016	0.064	0.144	0.576

There are 6 possible words Cavity ∨ toothache

$P(\text{Cavity} \vee \text{toothache}) = 0.108 + 0.012 + 0.072 + 0.008 + 0.016 + 0.064$

$= 0.28$

One particularly common task is to extract the distribution over some subset of variable or single variable.

eg:

marginal probability (or) unconditional probability

$$P(\text{cavity}) = 0.108 + 0.012 + 0.072 + 0.008 = 0.2$$

This is called marginalization. It is also can be written in the form of

$$P(Y) = \sum_{z \in Z} P(Y, z)$$

$$P(\text{cavity}) = \sum_{z \in \{\text{catch, toothache}\}} P(\text{Cavity}, z)$$

A variant of this rule involves conditional probability instead of joint probabilities, using the product rule

$$P(Y) = \sum_{z} P(Y|z)\, P(z)$$

This is called conditioning. and it provide a soln for all kind of probabilities.

$$P(\text{cavity} \mid \text{toothache}) = \frac{P(\text{cavity} \wedge \text{toothache})}{P(\text{toothache})}$$

$$= \frac{0.108 + 0.012}{0.108 + 0.012 + 0.016 + 0.064}$$

$$= 0.6$$

we can also compute there is no cavity given a toothache.

$$P(\neg \text{cavity} \mid \text{toothache}) = \frac{P(\neg \text{cavity} \wedge \text{toothache})}{P(\text{toothache})}$$

$$= \frac{0.016 + 0.064}{0.108 + 0.012 + 0.016 + 0.064}$$

$$= 0.4$$

Bayesian network:

To represent the dependencies among variables, a data structure called Bayesian networks.

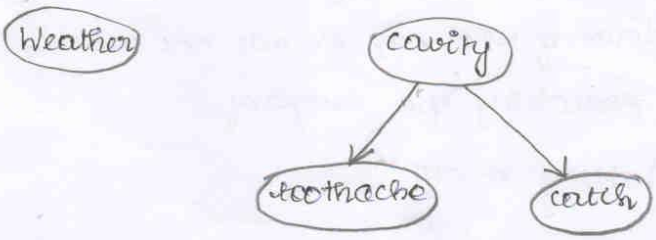
A Bayesian network is a directed graph in which each node is annotated with quantitative probability information.

1. Each node corresponds to a random variable, which may be discrete or continuous.

2. A set of directed links or arrows connects pair of nodes. If there is an arrow from node X to node Y, X is said to be a parent of Y. The graph has no directed cycles is a directed acyclic graph or DAG.

3. Each node X_i has a conditional probability distribution $P(X_i \mid \text{Parents}(X_i))$ that quantifies the effect of the parents on the node.

Eg for Simple Bayesian network:

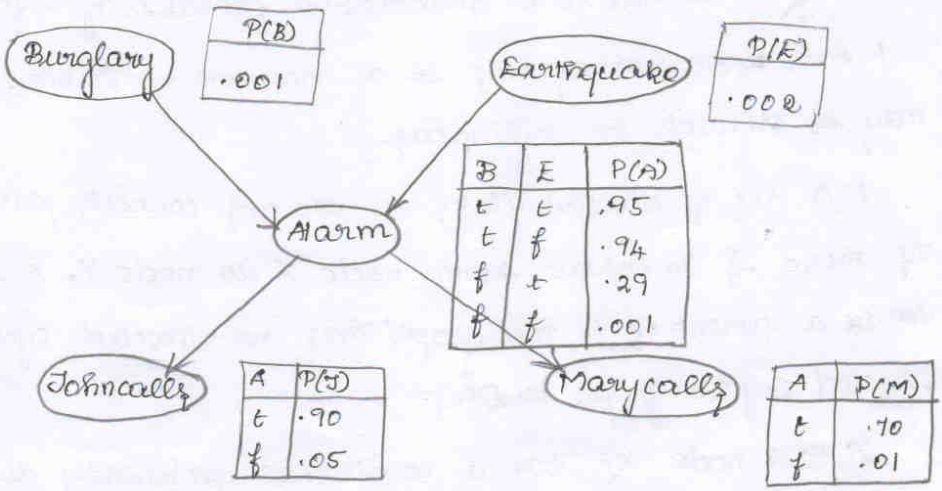


problem

Consider a simple ~~world~~, consists of variables Toothache, Cavity, catch and weather. Weather is independent of other three variables. Toothache and catch are conditional independence that could be observed by there is no link between them.

- conditional independence of Toothache and ~~cavity~~ catch, given cavity is indicated by the absense of a link between toothache and catch.

Eg: for Typical Bayesian network:



John nearly always calls when he hears the alarm, but sometimes confuses the telephone ringing with the alarm and calls then, too. Mary, on the other hand, likes rather loud music and often misses the alarm together. Given the evidence of who has or has not called, we have to estimate the probability of a burglary.

Conditional probability or CPT:

This form of table can be used for discrete variables, other representations, including those suitable for continuous variables.

Discrete random variables can be represented in the form of table known as conditional probability table.

Representing the full joint distribution:

A generic entry in the joint distribution is the probability of a conjunction of particular assignments to each variable, such as $P(X_1 = x_1 \wedge \ldots \wedge X_n = x_n)$. The formula for this entry is

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} \theta(x_i \mid \text{parents}(X_i)) \quad \rightarrow (1)$$

where parents (X_i) denotes the value of Parents (X_i) that appear in $x_1 \ldots x_n$. Thus each entry in the joint distribution is represented by the product of the appropriate element of the conditional probability tables (CPTs) in Bayesian network.

It is easy to prove that the parameters $\theta(X_i \mid \text{Parents}(X_i))$ are exactly the conditional probabilities $P(X_i \mid \text{Parents}(X_i))$ implied by joint distribution.

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid \text{parents}(X_i)). \quad \rightarrow (2).$$

Calculate the probability that the alarm has sounded, but neither a burglary nor an earthquake has occured, and both John and mary call.

$$P(j, m, a, \neg b, \neg e) = P(j \mid a)\, P(m \mid a)\, P(a \mid \neg b \wedge \neg e)\, P(\neg b)\, P(\neg e)$$

$$= 0.90 \times 0.70 \times 0.001 \times 0.999 \times 0.998.$$

$$= 0.000628.$$

If a Bayesian network is a representation of the joint distribution, then it too can be used to answer any query, by summing all the relevant joint entries.

A method for constructing Bayesian network:

$$P(x_1, \ldots x_n) = P(x_n \mid x_{n-1}, \ldots x_1)\, P(x_{n-1}, \ldots, x_1).$$

Repeating the process, reducing each conjunctive probability and a smaller conjunction.

$$P(x_1, \ldots x_n) = P(x_n \mid x_{n-1}, \ldots x_1)\, P(x_{n-1} \mid x_{n-2}, \ldots x_1) \ldots P(x_2 \mid x_1)\, P(x_1)$$

$$= \prod_{i=1}^{n} P(x_i \mid x_{i-1}, \ldots x_1)$$

This identity is called chain rule. It holds for any set of random variables. Comparing it with eqn (2), that the specification of the joint distribution is equivalent to the general assertion that, for every variable X_i, in network,

$$P(X_i \mid X_{i-1}, \ldots X_1) = P(X_i \mid \text{Parents}(X_i)) \rightarrow (3).$$

Condition:

i) Nodes: determining the set of variables that are required to model the domain.

ii) Links: there will be a conditional probability between successors and precedors nodes.

Bayesian Network :

A data structure to represent the dependencies among variables is called Bayesian Network.

A Bayesian Network is a directed graph in which each node is annoted with quantitative Probability information. The full specification are as follows :

1. Each node corresponds to a random variable, may be discrete or continuous.

2. A set of directed links or arrows connects pair of nodes. If there is an arrow from x to node y then it is said to be x is a parent of y. The graph has no directed cycles called directed acyclic Graph (DAG)

3. Each node x_i has a conditional Probability distribution $P(x_i \mid Parent(x_i))$ that quantifies the effect of the Parents on the node.

Approximate Inference in Bayesian Networks :

– Approximate Inference in Bayesian networks describes Randomized Sampling Algorithms called Monte carlo algorithm.

– Algorithm provides approximate answers whose accuracy depends on the number of samples generated.

– Two families of Algorithms are direct sampling and Markov chain planning.

Likelihood weighting :

Likelihood weighting avoids the inefficiency of a rejection sampling by generating only events that are consistent with the evidence e.

function LIKELIHOOD-WEIGHTING (X, e, bn, N) returns an estimate of P(X|e)

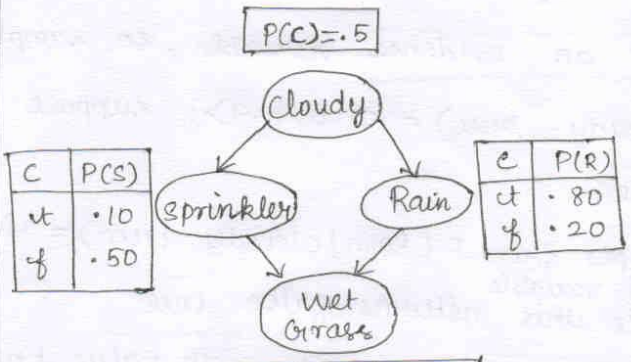
inputs : X, the query variable
e, observed values for variables E
bn, a bayesian network specifying joint distribution $P(x_1, \ldots x_n)$
N, the total number of samples to be generated.

local variables : W, a vector of weighted counts for each value of X, initially zero

```
for j=1 to N do
    X, w ← WEIGHTED-SAMPLE(bn, e)
    W[x] ← W[x] + w where x is the value of X in x
return NORMALIZE(W)

function WEIGHTED-SAMPLE(bn, e) returns an event and a weight
w ← 1; x ← an event with n elements initialized from e
for each variable Xi in X1, ... Xn do
    if Xi is an evidence variable with xi in e
        then w ← w × P(Xi = xi | parents(Xi))
        else x[i] ← a random sample from P(Xi | parents(Xi))
return x, w
```

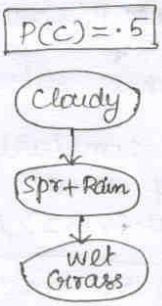
LIKELIHOOD - WEIGHTING fixes the value for the evidence variables E and samples only the nonevidence variables.



C	P(S)
t	.10
f	.50

C	P(R)
t	.80
f	.20

S	R	P(W)
t	t	.99
t	f	.90
f	t	.90
f	f	.00



C	P(S+R=x) tt	tf	ft	ff
t	.08	.02	.72	.18
f	.10	.40	.10	.40

S+R	P(W)
t t	.99
t f	.90
f t	.90
f f	.00

Let us apply the Algorithm to the NEtwork with the Query P(Rain | cloudy = true, WetGrass = true) and the ordering cloudy, sprinkler, Rain, WetGrass.

First, the weight w is set to 1.0. The following are the events :

1. cloudy is an evidence variable with value true;

then

$w \leftarrow w \times P(\text{cloudy} = \text{true}) = 0.5$

2. sprinkler is not an evidence variable, so sample from $P(\text{sprinkler} \mid \text{cloudy} = \text{true}) = \langle 1.0, 0.9 \rangle$; suppose this returns false

(*) 3. similarly sample from $P(\text{Rain} \mid \text{cloudy} = \text{true})$ where cloudy is an evidence variable $\langle 0.8, 0.2 \rangle$; suppose this returns true

4. WetGrass is an evidence variable with value true.

$\therefore w \leftarrow w \times P(\text{WetGrass} = \text{true} \mid \text{sprinkler} = \text{false}, \text{Rain} = \text{true}) = 0.45$

(*) 3. Similarly sample from $P(\text{Rain} \mid \text{cloudy} = \text{true}$, where the cloudy is an evidence variable $= \langle 0.8, 0.2 \rangle$;

suppose this returns true

1 Bayesian Network:

A Bayesian network is a directed graph in which each node is annotated with quantitative probability information. The full specification is as follows.

* Each node corresponds to a random variable which may be discrete or continuous.
* A set of directed links or arrows connects pair of nodes.
* Each node X_i has a conditional probability distribution.

Variable elimination algorithm:

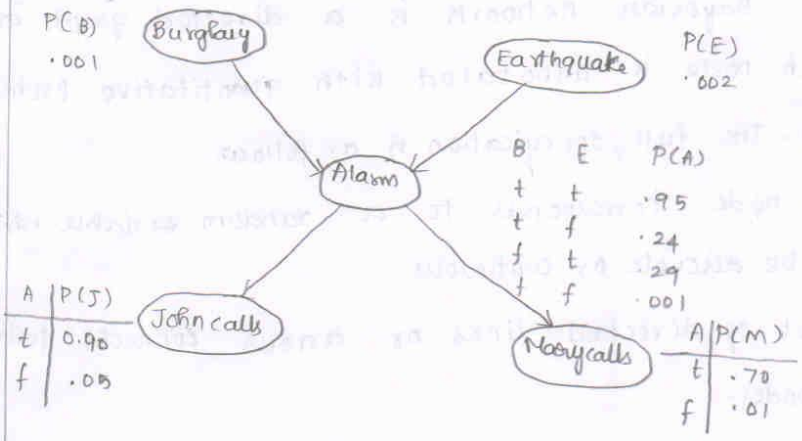
The elimination algorithm can be improved substantially by eliminating repeated calculation. This idea is simple to do to calculation once and save the result and this is a form of dynamic programming.

The variable elimination algorithm is the simplest elimination works by evaluating expression from right to left.

Such as the equation

$$P(b|j,m) = \alpha P(b) \sum_e P(e) \sum_a P(a|b,e) P(j|a) . P(m|a)$$

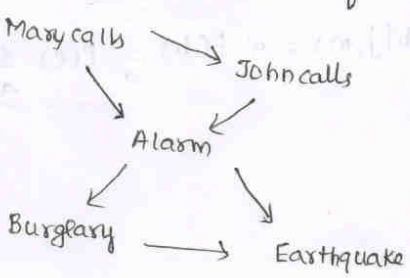
Example :



A Bayesian network for this domain is appears above. The network structure shows the burglary and earthquake directly affect the probability of the alarm's going off. But whether John and Mary call depends only on the alarm.

The network that represent our assumptions that they do not perceive burglaries directly they do not notice minor earthquake and they do not confer before calling.

The network structure depends on order of introduction in each network we have introduced nodes in top to bottom order



Let us illustrate this process for the burglary network. Then we evaluate the expression as.

$$P(B|j,m) = \alpha \underbrace{P(B)}_{F1} \sum_{e} \underbrace{P(e)}_{F2} \sum_{a} \underbrace{P(a|B,e)}_{F3} \underbrace{P(j|a)}_{F4} \underbrace{P(m|a)}_{F5}$$

consider f_4 & f_5

$$f_4(A) = \frac{P(j|a)}{P(j|\neg a)} = \begin{pmatrix} 0.90 \\ 0.05 \end{pmatrix}$$

$$f_5(A) = \frac{P(m|a)}{P(m|\neg a)} = \frac{0.70}{0.01}$$

$P(a|B,e) = 0.95$

And the above expression written as.

$$P(B|j,m) = \alpha f_1(B) \times \sum_{e} f_2(E) \times \sum_{a} f_3(A,B,E) \times f_4(A) \times f_5(A)$$

First we sum out A from the product of f_3, f_4 and f_5. This gives new factor $f_6(B,E)$ ranges over B & E

$$f_6(B,E) = \sum_{a} f_3(A,B,E) \times f_4(A) \times f_5(A)$$

Now we are left with the expression

$$P(B|j,m) = \alpha f_1(B) \times \sum_{e} f_2(E) \times f_6(B,E)$$

Now we sum f_2 and f_6 we get f_7

$$f_7(B) = \sum_{e} f_2(E) \times f_6(B,E)$$

$$\therefore P(B,j|m) = \alpha f_1(B) \times f_7(B)$$

Examining this sequence, we see the two basic computational operations are required they are point wise product of a pair of factors and summing out a variable from a product of factor.

Algorithm :

Variable elimination algorithm for inference in Bayesian network.

function ELIMINATION-ASK (X, e, bn) returns a distribution over X.

inputs : X, the query variable

e, observed values for variable E

bn, a bayesian network specifying joint distribution $P(X_1 \ldots X_n)$

factors ← [ ]

For each var in ORDER (bn, VARS) do

Factors ← [MAKE-FACTOR (var, e) | factors]

if var is a hidden variable then

factors ← SOM-OUT (var, factors)

return NORMALIZE (POINTWISE-PRODUCT (factors)).

1) Approximate Inference in Bayesian Networks :-

* Bayesian Network is a data structure that represents the dependencies among the variables.

* A bayesian network is a directed graph in which each node is annotated with quantitative probability information.

* In a bayesian network,

i) Each node corresponds to a random variable, which may be discrete or continuous.

ii) The arrow connects the pair of nodes. If there is an arrow from node X to node Y, then X is said to be the parent of Y. The graph has no directed cycles.

iii) Each node X_i has a conditional probability distribution $P(X_i \mid Parents(X_i))$ that quantifies the effect of the parents on the node.

* Approximate Inference provide approximate answers whose accuracy depends on the number of samples generated.

* Four mechanisms used for approximate inference in Bayesian Networks are (i) Direct Sampling. (ii) Rejection Sampling (iii) Markov chain Sampling (iv) Likelihood weighting

Direct Sampling Methods :-

* The primitive element in any sampling algorithm is the generation of samples from a known

probability distribution.

* The simplest kind of random sampling process for Bayesian networks generates events from a network that has no evidence associated with it.

* The idea is to sample each variable in the topological order.

* For example, consider the following ordering

[cloudy, Sprinkler, Rain, Wetgrass]

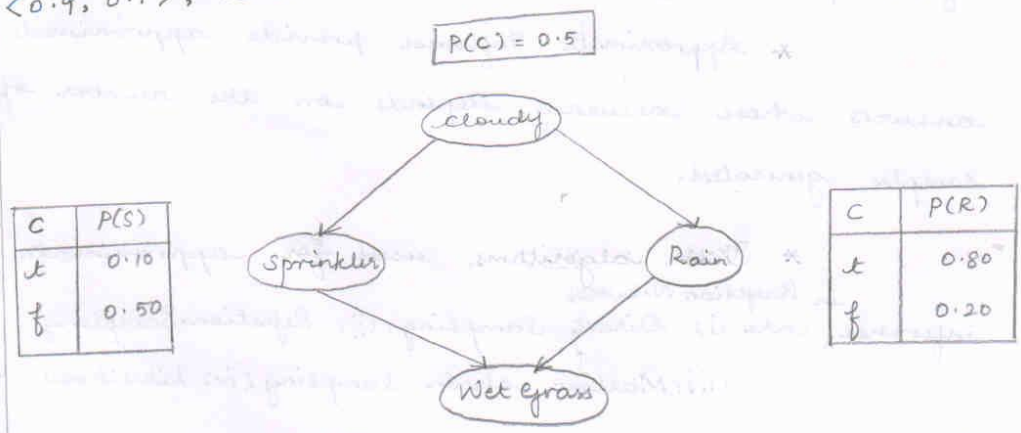
The given samples are,

i) Sample from P(cloudy) = <0.5, 0.5>, value is true.

ii) Sample from P(Sprinkler | cloudy = true) = <0.1, 0.9>, value is false.

iii) Sample from P(Rain | cloudy = true) = <0.8, 0.2>, value is false.

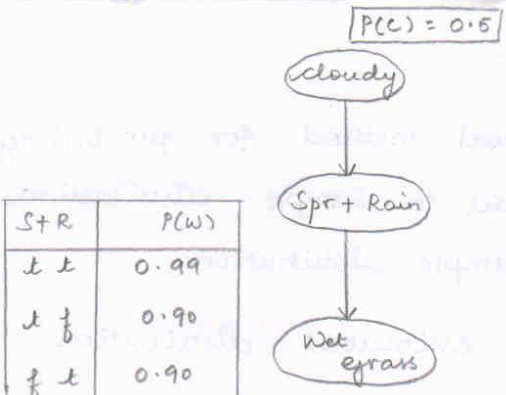
iv) Sample from P(Wetgrass | Sprinkler = false, Rain = true) = <0.9, 0.1>, value is true.



C	P(S)
t	0.10
f	0.50

C	P(R)
t	0.80
f	0.20

S	R	P(W)
t	t	0.99
t	f	0.90
f	t	0.90
f	f	0.00



S+R	P(W)
t t	0.99
t f	0.90
f t	0.90
f f	0.00

C	P(S+R = x)			
	tt	tf	ft	ff
t	0.80	0.02	0.72	0.18
f	0.10	0.40	0.10	0.40

4.25

* To find the P(Wetgrass | Rain, Sprinkler)

Here, the evidence are Rain and Sprinkler.

The hidden variable is Cloudy.

* In this case, prior sample returns the event [True, False, True, True]

SAMPLING ALGORITHM :-

function PRIOR-SAMPLE(bn) returns an event sampled from the prior specified by bn.

inputs : bn, a Bayesian network specifying joint distribution $P(X_1, \ldots, X_n)$

x ← an event with n elements.

foreach variable X_i in $X_1, \ldots, X_n$ do

x[i] ← a random sample from $P(X_i \mid Parents(X_i))$

return x

Rejection Sampling:

It is a general method for producing samples from a hard-to sample distribution given an easy-to sample distribution

Let $P(x|e)$ be the estimated distribution from that we have

$$P(x|e) = \alpha N_{PS}(x,e) = \frac{N_{PS}(x,e)}{N_{PS}(e)}$$

e - evident true.

$$\Rightarrow P(x|e) \approx \frac{P(x,e)}{P(e)} = P(x|e)$$

Let us assume that we have 100 samples.

To Find P(Rain | Sprinkler = true)

Suppose out of 100 samples 73 have Sprinkler = False & are Rejected. 27 have Spiinkler = true.

Out of 27, 8 have Rain = true & 19 have Rain = false

$\therefore$ P(Rain | Sprinkler = true) $\simeq$ NORMALIZE (<8, 19>) α 8/27

$= \langle 0.296, 0.704 \rangle$

Algorithm:

function REJECTION-SAMPLING(X, e, bn, N) returns

P.No - 542.

1. Why we move to clustering algorithm in Exact inference in Bayesian networks?

4.27

Bayesian Networks:

Bayesian Networks is a data structure to represent the dependencies among variables.

Bayesian Networks can represent ~~essentially any~~ by full joint probability distribution.

A Bayesian Network is a directed graph, it contains the following information.

* Each node corresponds to a random variable which may be discrete or continuous.

* A set of directed link or arrows connected pairs of nodes.

* Each node X_i has a conditional probability distribution $P(X_i \mid Parents(X_i))$ that quantifies the effect of the parents on the node.

Exact Inference in Bayesian Networks:

The basic task for any probabilistic inference system is to compute the posterior probability distribution for a set of query variables, event a some assignment of values to a set of evidence variable. It take one queries at a time.

X denotes the query variable, E denotes the set of evidence variables $E_1 \cdots E_m$ and e is a particular observed event, Y will denotes the non evidence, nonquery variables $Y_1 \cdots Y_l$ called as hidden variables.

Exact algorithm for computing postarior probabilities and will consider the complexity of this task.

It covers methods for approximate inference. Here we discuss one of the method. That is clustering method.

Clustering algorithm:

Drawback of the complexity of exact inference:

It avoids the repeated computations as well as dropping irrelevant variables. The time and space requirements of variable elimination are dominated by the size of the largest factor constructed during the operations of the algorithm.

Eg: There is at most one undirected path between any two nodes in the network. It is called as singly connected networks (or) polytrees. The particular property of this problem is

[The time and space complexity of exact inference in polytrees is linear in the size of the network. Here the size is defined as the number of CPT entries. The complexity will also be linear in the number of nodes]

For multiply connected networks, variable elimination can have exponential time and space complexity in the worst case.

So we move to the clustering algorithm.

Clustering algorithm: → Multiple Path to single path by Combining Nodes.

The variable elimination algorithm is simple and efficient for answering individual queries.

If we want to compute posterior probabilities for all the variables in a network, it can be left efficient.

For example in a polytree Network.

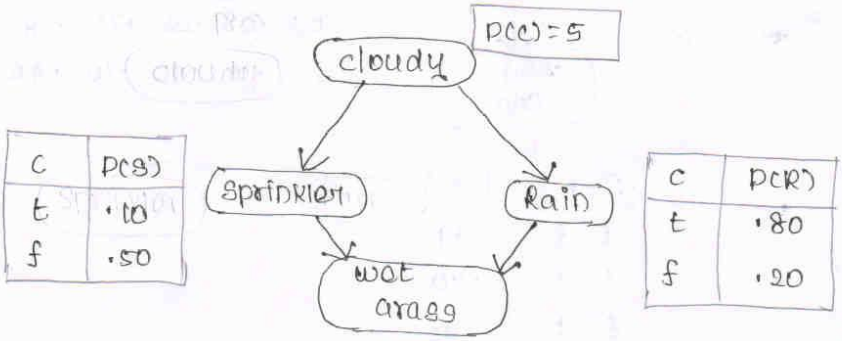
one would need to issue O(n) queries costing O(n) each, for a total of $O(n^2)$ time.

Using clustering algorithm it also known as joint tree algorithm.

The time can be reduced to O(n), this algorithms are widely used in commercial Bayesian network tools.

The basic idea of clustering is to join individual nodes of the network to form cluster nodes, that the resulting network is a polytree network.

For example, multiple connected network.



C	P(S)
t	.10
f	.50

C	P(R)
t	.80
f	.20

S	R	P(W)
t	t	.99
t	f	.90
f	t	.90
f	f	.00

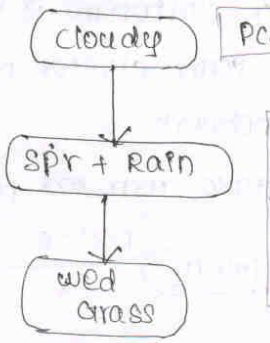
It is a multiply connected network with conditional probability tables. Here two variables are used as a Boolean variables to check the conditions if it is true (or) false.

Then we convert this conditional probability tables to the clustering equivalent of the multiply connected network by it converted into a polytree by combining the sprinkler and Rain node into a cluster node called as Sprinkler + Rain.

Sprinkler and Rain are called as two Boolean nodes, the two Boolean nodes are replaced by a one meganode called as Sprinkler + Rain.

It takes on four values as TT, TF, FT and FF.

The clustered equivalent of the multiply connected network is given bellow.



P(C) = 5

	P(S+R = x)			
C	tt	tf	ft	ff
t	.08	.02	.72	.18
f	.10	.40	.10	.40

S+R	P(W)
t t	.99
t f	.90
f t	.90
f f	.00

The maganode (Spr+Rain) has only one parent, the Boolean variable cloudy, so there are two conditioning causes.

Once the network is in polytree form, a special-purpose inference algorithm is applied. Essentially the algorithm is a form of constraint propagation where the constraints ensure that neighboring clusters agree on the posterior probability of any variables that they have a common.

This algorithm is able to compute posterior probabilities for all the nonevidence nodes in the network in time $O(n)$, where n is the size of the modified network.

The NP-hardness of the problem has not disappeared, if a network requires exponential time and space with variable elimination.

The clustering network require exponential time and space to construct.

Decision tree :

A decision tree represents a function that takes a input as vector of attribute values and returns a decision a single output value. It can be discrete or continuous. Input will be classified as true or false.

As an example we will build a decision tree to decide whether to wait for a table at a restaurant. The aim here is to learn a definition for the goal predicate willwait.

Attributes :

1. Alternate : whethere there is suitable alternate restaurant nearby.
2. Bar : It has a comfortable bar area to wait in
3. Fri/sat : true on Fridays and saturdays.
4. Hungry : whether we are hungry
5. Patrons : How many people are in restaurent
6. Price : the restaurent price range
7. Raining : Whether it is raining outside
8. Reservation : whether we made a reservation
9. Type : the kind of restaurent (French, Italian, Thai, burger)
10. Wait Estimate : the wait estimated by the host.

Expressiveness of decision trees :

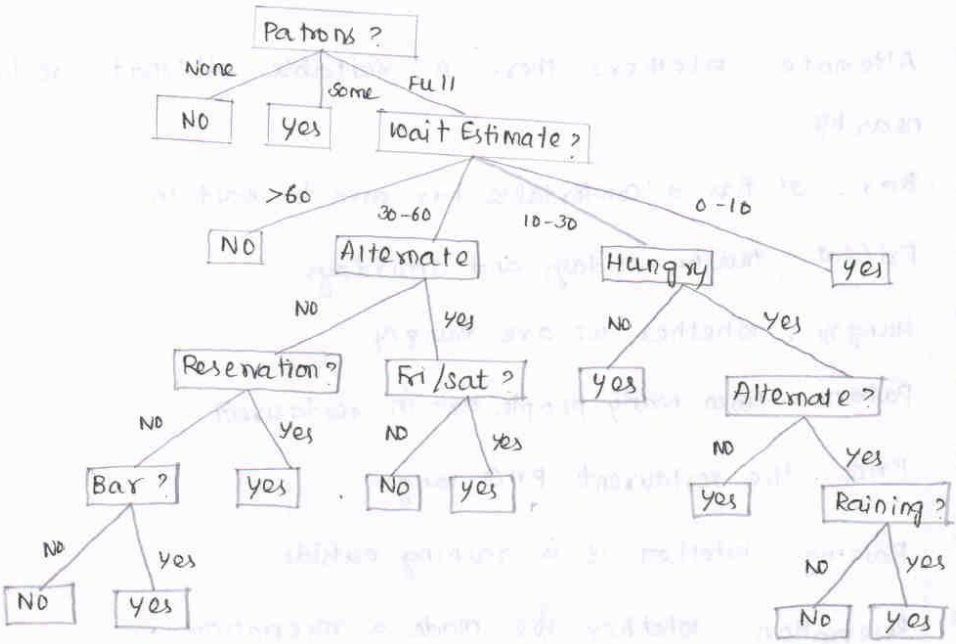
A boolean decision tree is logically equivalent to the assertion that the goal attribute is true if and only if the input attribute satisfy one of the paths leading to a leaf with value true in propositional logic

Goal $\Leftrightarrow$ (Path$_1$ $\vee$ Path$_2$ $\vee$ ...)

Each path is a conjunction of attribute value tests required to follow that path.

Decision tree :

A decision tree for deciding whether to wait for a table.



The boolean decision tree consists of an (x, y) pair where x is a vector of values for the input attributes and y is a single boolean output value.

Inference :

Example	Input Attribute										Goal
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	will wait
X_1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0-10	y_1 = Yes
X_2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	y_2 = No
X_3	No	Yes	No	No	Some	$	No	No	Burger	0-10	y_3 = Yes
X_4	Yes	No	Yes	Yes	Full	$	Yes	No	Thai	10-30	y_4 = Yes
X_5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	y_5 = No
X_6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0-10	y_6 = Yes
X_7	No	Yes	No	No	None	$	Yes	No	Burger	0-10	y_7 = No
X_8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0-10	y_8 = Yes
X_9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	y_9 = No
X_{10}	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	y_{10} = No
X_{11}	No	No	No	No	None	$	No	No	Thai	0-10	y_{11} = No
X_{12}	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	y_{12} = Yes

There are four cases to concide for these recursive branches :

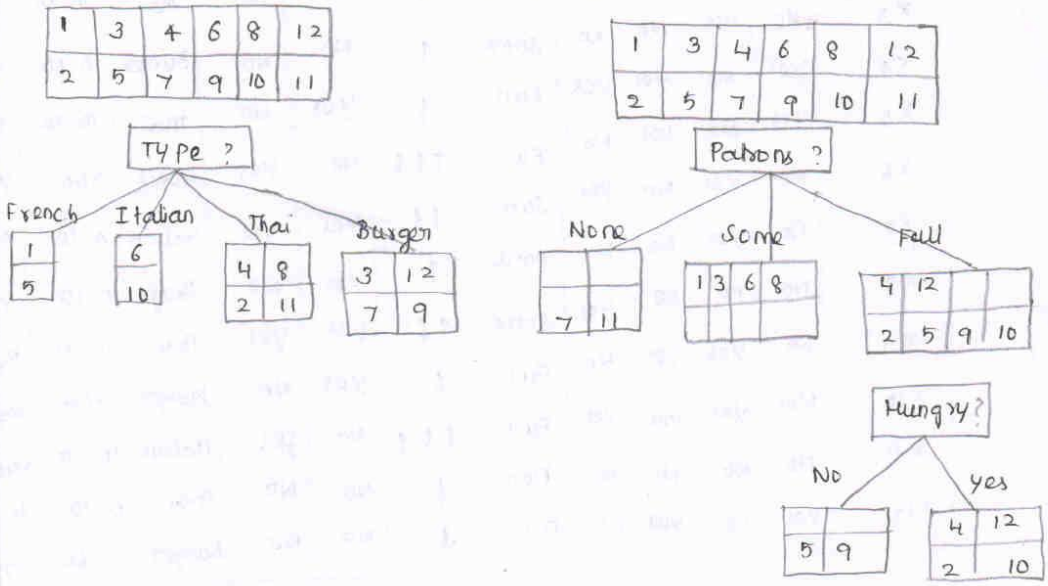
1. If the remaining examples are all positive or negative we can answer yes or No.

2. If there are some positive and negative examples then choose the best attribute to split them.

3. If there are no examples left it means that no example has been observed for this combination of attribute value and return a default value

4. If there are no attribute left but both positive and negative examples it means that the example have exactly the same description but different classification

Splitting the example by testing an attributes:



Algorithm:

Function DECISION-TREE LEARNING (examples, attributes, default) returns a tree

if examples is empty then return default (Parent_examples)

else if all examples have the same classification then return the classification.

else if attribute is empty then return majority_value (examples)

else

Best ← choose-attributes (attributes, examples)

tree ← a new decision tree with root test Best

M ← Majority-value (examples)

for each value v_K of Best do

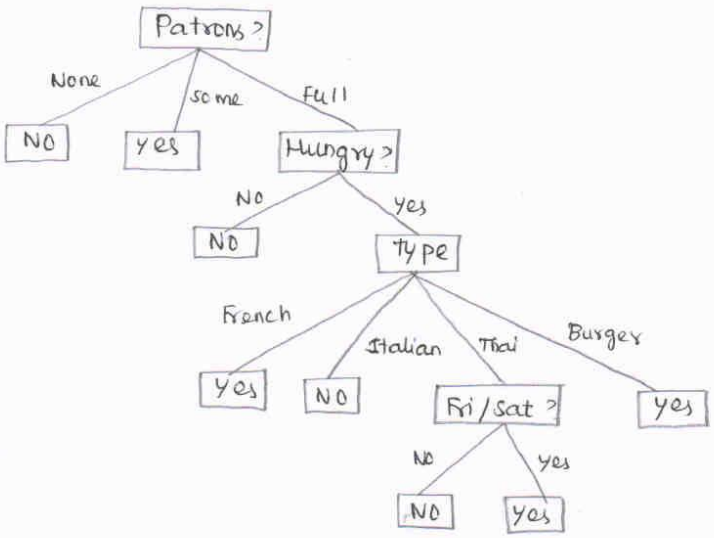
exs ← {e : e ∈ examples and e.Best = v_K}

subtree ← Decision Tree Learning (exs, attributes − Best, examples)

add a branch to tree with label (Best = v_K) and subtree subtree

return tree

The decision tree induced from the 12-example training set :



Backpropagation Learning:

Neural Network:

A neural network is a collection of units connected together. the properties of the network are determined by its topology and the properties of the "neurons". It is implements a linear classifier of the kind described in the preceding section.

Neural network structures:

Neural networks are composed of nodes (or) units connected by directed links.

A link from unit i to unit j serves to propagate the activation a_i from i to j. Each link also has a numeric weight $w_{i,j}$ associated with it.

which determines the strength and sign of the connection. Each units j first computes a weighted sum of its inputs.

There are two fundamentally distinct ways.

feed-forward networks has connections only one in one directions - it forms directed acyclic graph. A feed-forward network represents a function of its current input. it has no internal state other than the weights.

Recurrent Network has its outputs back into its own inputs.

It have one (or) more layers of hidden units that are learning problems with a single output variable. hidden units have not connected to the outputs of the network.

Single-layer feed-forward neural networks:

All the inputs connected directly to the output is called a single-layer neural network (or) a perceptron network.

It depending on the type of the activation function used, it processes will be either the perceptron learning rule (or) gradient descent rule for the logistic regression.

Disadv: the solution problem is easily represented as a decision tree, but is not be linearly separable.

Multilayer-feed-forward neural networks:

It connecting large numbers of units into networks of arbitrary depth. The problem was that nobody knew how to train such networks.

It composed of nested nonlinear soft threshold functions, neural networks as a tool for doing nonlinear regression.

For any particular network structure, it is harder to characterize exactly which functions can be represented and which one cannot.

Learning in multilayer networks:

The interaction among the learning problems when the network has multiple outputs.

The major complication comes from the addition of hidden layers to the network. The error at the output layer is clear, the error at the hidden layers.

The training data donot say what value the hidden node should have, it turn out that we can back propagate.

The error from the output layer to the hidden layer, the back propagation process emerges directly from a derivation of the overall error gradient.

At the output layer, the weight update rule is identical, the weight update rule becomes.

$$W_{j,k} \leftarrow W_{j,k} + \alpha \times a_j \times \Delta_k.$$

To update the connections b/w the input unit and the hidden units, we need to define the quantity to the error term from for the output nodes.

The Δ_k values are divided according to the strength of the connection b/w the hidden node and the output node are propagated back to provide the Δ_j values for the hidden layer.

The propagation rule for the Δ values,

$$\Delta_j = g'(in_j) \sum_k w_{j,k} \Delta_k .$$

The weight-update rule for the weights b/w the input and hidden layer is essentially identical to the update rule for the output layer.

$$w_{i,j} \leftarrow w_{i,j} + \alpha * a_i \times \Delta_j$$

The back propagation process can be summarized as follows.

Compute the Δ values for the output units, using the observed error.

Starting with output layer, repeat the following for each layer in the network, until the earliest hidden layer is reached

=> propagate the Δ values back to the previous layer.

=> Update the weights between the two layers.

Algorithm:

The algorithm => back propagation algorithm for learning in multilayer networks.

Refer page No: 745 => figure 24.