Mini-Max Algorithm in Artificial Intelligence

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.
-

Pseudo-code for MinMax Algorithm:

```
1. function minimax(node, depth, maximizingPlayer) is
2. if depth ==0 or node is a terminal node then
3. return static evaluation of node
4. if MaximizingPlayer then      // for Maximizer Player
5. maxEva= -infinity
6.  for each child of node do
7.  eva= minimax(child, depth-1, false)
8. maxEva= max(maxEva,eva)       //gives Maximum of the values
9. return maxEva
10.
11. else                         // for Minimizer player
12.  minEva= +infinity
13.  for each child of node do
14.  eva= minimax(child, depth-1, true)
15.  minEva= min(minEva, eva)       //gives minimum of the values
16.  return minEva
```

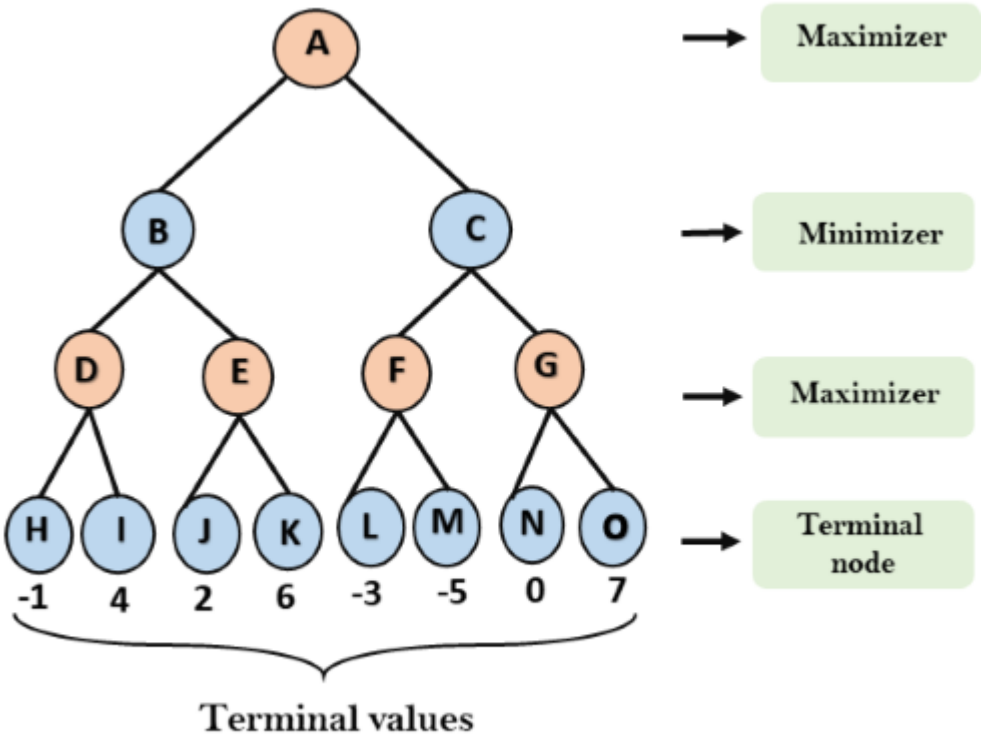
Initial call:

Minimax(node, 3, true)

Working of Min-Max Algorithm:

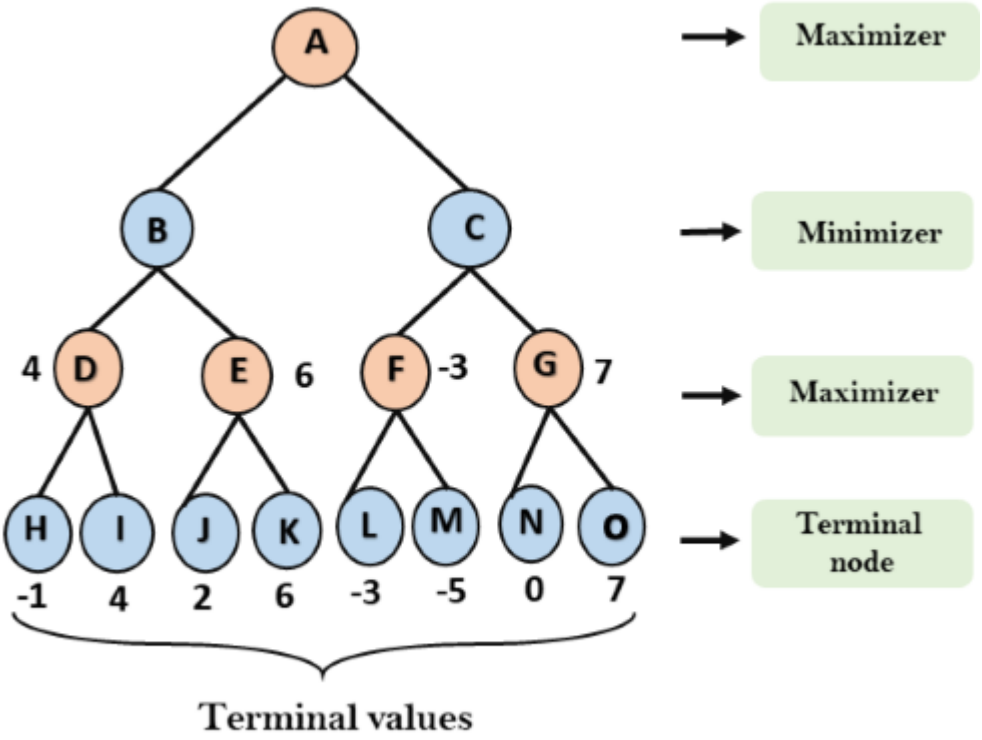
- The working of the minimax algorithm can be easily described using an example. Below we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



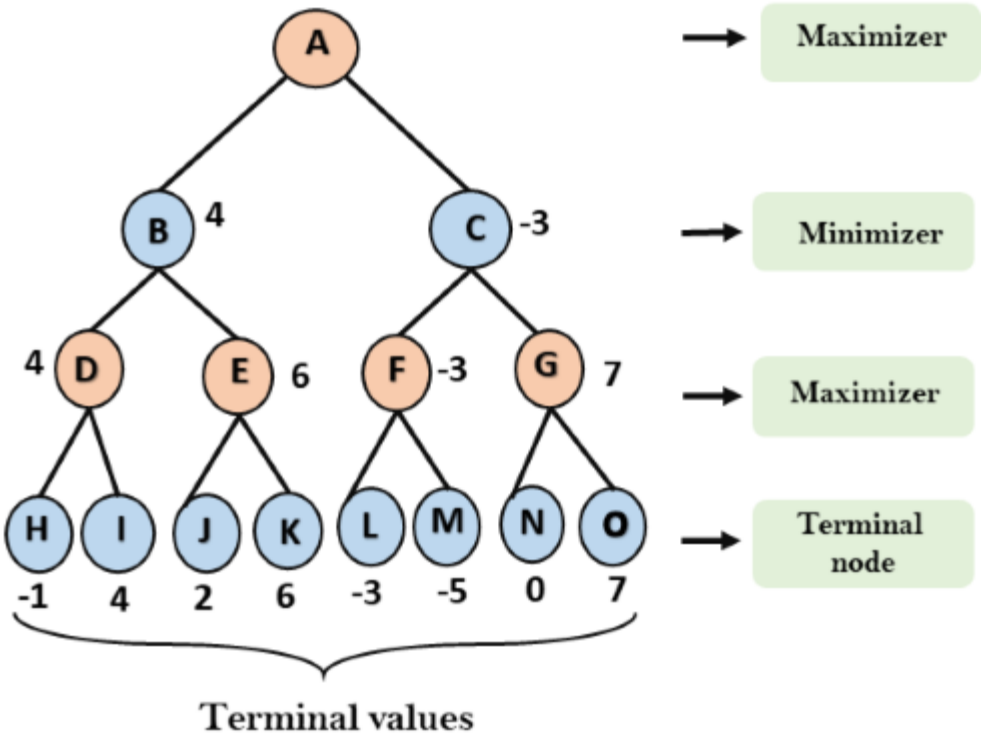
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- For node D max(-1,- -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



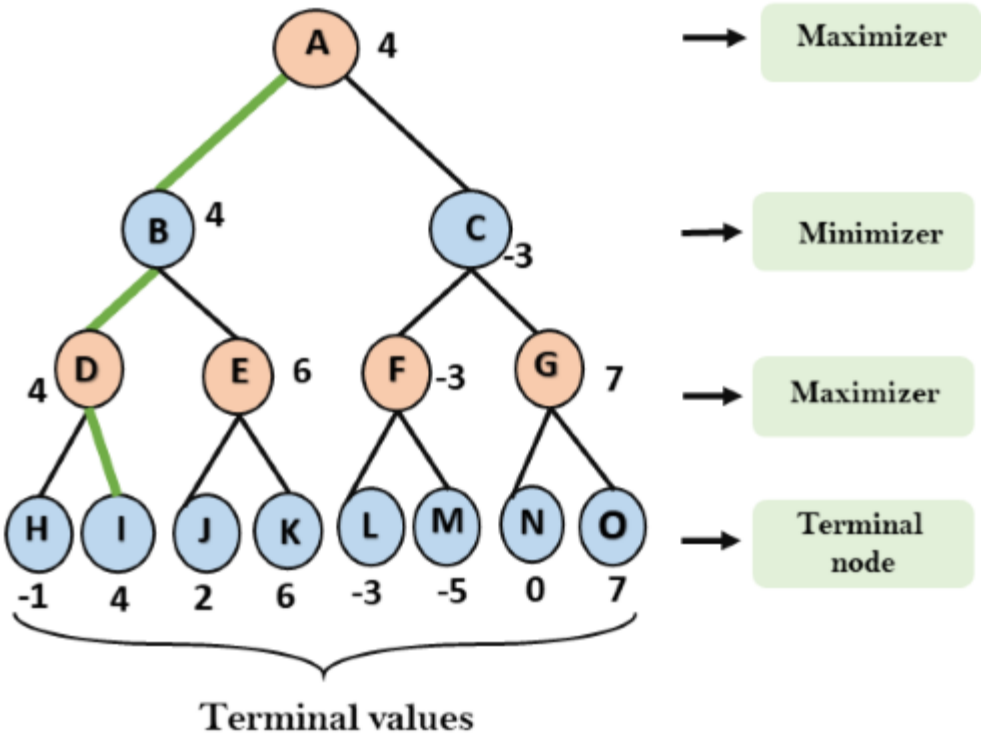
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A max(4, -3)= 4



That was the complete workflow of the mini-max two player game.

Properties of Mini-Max algorithm:

- Complete- Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- Optimal- Min-Max algorithm is optimal if both opponents are playing optimally.
- Time complexity- As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is O(bm), where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- Space Complexity- Space complexity of Mini-max algorithm is also similar to DFS which is O(bm).

Limitation of the minimax Algorithm:

The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the player has lots of choices to decide. This limitation of the minimax algorithm can be improved from alpha-beta pruning

PREDICATE LOGIC OR FIRST ORDER LOGIC

FOL or predicate logic is a generalization of propositional logic that allows us to express and infer arguments in infinite models like

- All men are mortal
- Some birds cannot fly

Predicates are like functions except that their return type is true or false.

A predicate with no variable is a proposition.

	Propositional Logic	Predicate Logic
1	Propositional logic is the logic that deals with a collection of declarative statements which have a truth value, true or false.	Predicate logic is an expression consisting of variables with a specified domain. It consists of objects, relations and functions between the objects.
2	It is the basic and most widely used logic. Also known as Boolean logic.	It is an extension of propositional logic covering predicates and quantification.
3	A proposition has a specific truth value, either true or false.	A predicate's truth value depends on the variables' value.
4	Scope analysis is not done in propositional logic.	Predicate logic helps analyze the scope of the subject over the predicate. There are three quantifiers : Universal Quantifier (∀) depicts for all, Existential Quantifier (∃) depicting there exists some and Uniqueness Quantifier (∃!) depicting exactly one.
5	Propositions are combined with Logical Operators or Logical Connectives like Negation(¬), Disjunction(∧), Conjunction(∨), Exclusive OR(⊕), Implication(⇒), Bi-Conditional or Double Implication(⇔).	Predicate Logic adds by introducing quantifiers to the existing proposition.
6	It is a more generalized representation.	It is a more specialized representation.
7	It cannot deal with sets of entities.	It can deal with set of entities with the help of quantifiers.

Predicates

- A clause has a subject and a predicate.
- To be a sentence (an independent clause), there must be a subject and a predicate, and it needs to be a complete thought.
- A simple predicate is a verb; a complete predicate is everything that's not the subject.

Types of Predicates

- Felix *laughed*.
- Winnie *will sing.*
- The grass *is always greener on the other side*.
- Sandy *prefers to run first and then eat breakfast afterward.*

Finding the Predicate

- *After the long hike up the mountain*, the tour group *rested and took in the views.*

Examples of Predicates

1. Time *flies*.
2. We *will try*.
3. The Johnsons *have returned*.
4. Bobo *has never driven before*.
5. We *will try harder next time*.
6. Hummingbirds *sing with their tail feathers*.
7. Pedro *has not returned from the store*.
8. My brother *flew a helicopter in Iraq*.
9. My mother *took our dog to the vet for its shots*.
10. Our school cafeteria *always smelled like stale cheese and dirty socks*.

Syntax of First-Order Logic

- Constants KingJohn, 2, …
- Predicates/Relation Brother, >, …
- Functions Sqrt, LeftArmOf, …
- Variables x, y, a, b, …
- Connectives ∧ ∨ ¬ ⇒ ⇔
- Equality =
- Quantifiers

Components of First-Order Logic

- **Term**
    - Constant, e.g. Red

- Function of constant, e.g. Color(Block1)

- **Atomic Sentence**
  - Predicate relating objects (no variable)
    - ➢ Brother (John, Richard)
    - ➢ Married (Mother(John), Father(John))
- **Complex Sentences**
  - Atomic sentences + logical connectives
    - Brother (John, Richard) ∧¬Brother (John, Father(John))
- **Quantifiers**
  - Each quantifier defines a variable for the duration of the following expression, and indicates the truth of the expression…
- **Universal quantifier "for all"** ∀
  - The expression is true for every possible value of the variable
- **Existential quantifier "there exists"** ∃
  - The expression is true for at least one value of the variable

Examples

1. Some dogs bark.

∃x dog(x) ∧ bark(x)

2. All dogs have four legs.

∀x(dog(x)?has_four_legs(x))

(or)

∀x(dog(x)?legs(x,4))

3. All barking dogs are irritating

∀x(dog(x) ∧barking(x)?irritating(x))

4. No dogs purr.

¬∃x (dog(x) ∧ purr(x))

5. Fathers are male parents with children.

∀x(father(x)?male(x) ∧ haschildren(x))

6. Students are people who are enrolled in courses.

∀x(student(x)?enro lled(x, courses))

First order Logic

(a) Marcus was a man.
(b) Marcus was a Roman.
(c) All men are people.
(d) Caesar was a ruler.
(e) All Romans were either loyal to Caesar or hated him (or both).
(f) Everyone is loyal to someone.
(g) People only try to assassinate rulers they are not loyal to.
(h) Marcus tried to assassinate Caesar.

(a) man(marcus)
(b) roman(marcus)
(c) ∀X. man(X) → person(X)
(d) ruler(caesar)
(e) ∀X. roman(x) → loyal(X,caesar) ∨ hate(X,caesar)
(f) ∀X∃Y. loyal(X,Y)
(g) ∀X∀Y. person(X) ∧ ruler(Y) tryassasin(X,Y) → ¬loyal(X,Y)
(h) tryassasin(marcus,caesar)

ALGORITHM: RESOLUTION-PREDICATE LOGIC

1. Convert all the statements of F to clause form.

2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.

3. Repeat until either a contradiction is found or no progress can be made

(a) Select two clauses. Call these the parent clauses.

(b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed

and with the following exceptions: If there is one pair of literals T1 and ¬T2 such that one of the parent clause contains T2 and the other contains t1 and if t1 and t2 are unifiable, then neither T1 nor T2 should appear in the resolvent. We call T1 and T2 as complementary literals. Use the substitution produced by the unification to create the resolvent. If there is more than pair of complementary literals, only one pair should be omitted from the resolvent.

(c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

There exists a procedure for making the choice that can speed up the process considerably

- Only resolve pairs of clauses that contain complementary literals
- Eliminate certain clauses as soon as they are generated so that they cannot participate in later resolutions.
- Whenever possible resolve either with one of the clauses that is part of the statement we are trying to refute or with clause generated by a resolution with such clause. This is called set of support strategy
- Whenever possible resolve with clauses that have a single literal. Such resolution generate new clauses with fewer literals. This is called unit preference strategy.

Problem 1

1. All people who are graduating are happy.

2. All happy people smile.

3. Someone is graduating.

4. Conclusion: Is someone smiling?

Solution:

Convert in to predicate Logic

1. ∀x[graduating(x)[?]happy(x)

2. ∀x(happy(x)[?]smile(x)

3. ∃x graduating(x)

4. ∃x smile(x)

Convert to clausal form

(i) Eliminate the [?] sign

1. ∀x¬graduating(x) ∨happy(x)

2. ∀x¬happy(x) ∨smile(x)

3. ∃x graduating(x)

4. ¬∃x smile(x)

(ii) Reduce the scope of negation

1. ∀x¬graduating(x) ∨happy(x)

2. ∀x¬happy(x) ∨smile(x)

3. ∃x graduating(x)

4. ∀x ¬ smile(x)

(iii) Standardize variables apart

1. ∀x¬graduating(x) ∨happy(x)

2. ∀y¬happy(y) ∨smile(y)

3. ∃x graduating(z)

4. ∀w ¬ smile(w)

(iv) Move all quantifiers to the left

1. ∀x¬graduating(x) ∨happy(x)

2. ∀y¬happy(y) ∨smile(y)

3. ∃x graduating(z)

4. ∀w ¬ smile(w)

(v) Eliminate ∃(Skolemization)

Skolemization: remove existential quantifiers by introducing new function symbols

1. ∀x¬graduating(x) ∨happy(x)

2. ∀x¬happy(y) ∨smile(y)

3. graduating(name1)

4. ∀w ¬ smile(w)

(vi) Eliminate ∀

1. ¬graduating(x) ∨happy(x)

2. ¬happy(y) ∨smile(y)

3. graduating(name1)

4. ¬ smile(w)

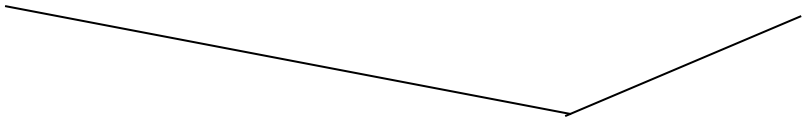
(vii) Convert to conjunct of disjuncts form.

All ready it is in conjunct of disjuncts form only

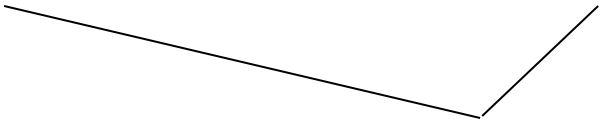
(viii) Make each conjunct a separate clause.

(ix) Standardize variables apart again.

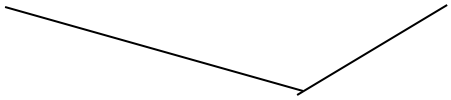
¬happy(y) ∨smile(y) ¬ smile(w)



¬graduating(x) ∨happy(x) ¬happy(y)



. graduating(name1) ¬graduating(x)



None

Thus, we proved someone is smiling.

Note : Apply Unification or substitution algorithm

<u>Example 2 :</u>

(a) Marcus was a man.

(b) Marcus was a Roman.

(c) All men are people.

(d) Caesar was a ruler.

(e) All Romans were either loyal to Caesar or hated him (or both).

(f) Everyone is loyal to someone.

(g) People only try to assassinate rulers they are not loyal to.

(h) Marcus tried to assassinate Caesar

First order Logic

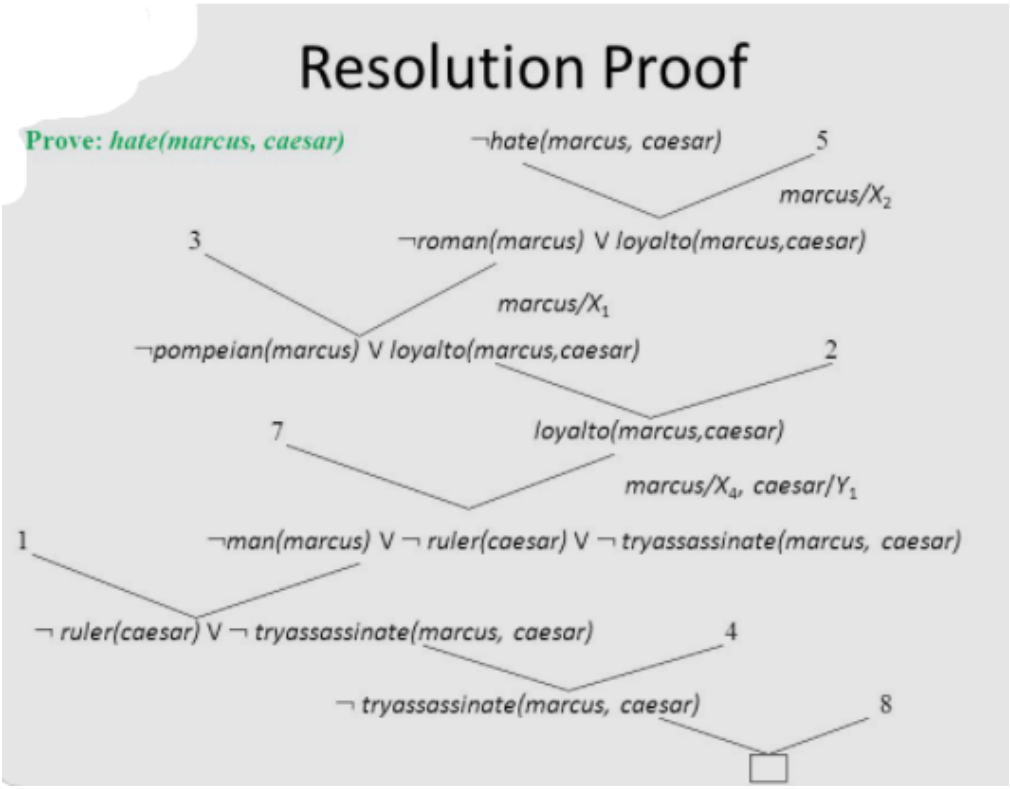
(a) Marcus was a man.
(b) Marcus was a Roman.
(c) All men are people.
(d) Caesar was a ruler.
(e) All Romans were either loyal to Caesar or hated him (or both).
(f) Everyone is loyal to someone.
(g) People only try to assassinate rulers they are not loyal to.
(h) Marcus tried to assassinate Caesar.

(a) man(marcus)
(b) roman(marcus)
(c) $\forall X.\ man(X) \rightarrow person(X)$
(d) ruler(caesar)
(e) $\forall X.\ roman(x) \rightarrow loyal(X,caesar) \vee hate(X,caesar)$
(f) $\forall X \exists Y.\ loyal(X,Y)$
(g) $\forall X \forall Y.\ person(X) \wedge ruler(Y)\ tryassasin(X,Y) \rightarrow \neg loyal(X,Y)$
(h) tryassasin(marcus,caesar)

Convert to Clausal Form

(a) man(marcus)
(b) roman(marcus)
(c) $\forall X.\ man(X) \rightarrow person(X)$
$(\neg man(X), person(X))$
(d) ruler(caesar)
(e) $\forall X.\ roman(X) \rightarrow loyal(X,caesar) \vee hate(X,caesar)$
$(\neg roman(X), loyal(X,caesar), hate(X,caesar))$
(f) $\forall X \exists Y.\ loyal(X,Y)$
$(loyal(X,f(X))$
(g) $\forall X \forall Y.\ person(X) \wedge ruler(Y) \wedge tryassasin(X,Y) \rightarrow \neg loyal(X,Y)$
$(\neg person(X), \neg ruler(Y), \neg tryassasin(X,Y), \neg loyal(X,Y))$
(h) tryassasin(marcus,caesar)

Resolution Proof Example.



UNIFICATION ALGORITHM

When attempting to match 2 literals, all substitutions must be made to the entire literal. There may be many substitutions that unify 2 literals, the most general unifier is always desired

1. Initial predicate symbols must match.

2. For each pair of predicate arguments:

– different constants cannot match.

– a variable may be replaced by a constant.

– a variable may be replaced by another variable.

– a variable may be replaced by a function as long as the function does not contain an instance of the variable

Algorithm :Unify(L1,L2)

1. If L1 or L2 are both variables or constants, then:

(a) If L1 and L2 are identical, then return NIL.

(b) Else if L1 is a variable, then if L1 occurs in L2 the return {FAIL}, else return (L2/L1).

(c) Else if L2 is a variable, then if L2 occurs in L1 the return {FAIL}, else return (L1/L2).

(d) Else return {FAIL}

2. If the initial predicate symbols in L1 and L2 are not identical, the return {FAIL}.

3. If L1 and L2 have a different number of arguments, then return {FAIL}.

4. Set SUBST to NIL.

5. For i ? 1 to number of arguments in L1:

(a) Call Unify with the i^{th} argument of L1 and the i^{th} argument of L2, putting result in S.

(b)If s contains FAIL then return {FAIL}

(c) If S is not equal to NIL then:

(i) Apply S to the remainder of both L1 and L2

(ii) SUBST=APPEND(S,SUBST).

6. Return SUBST.

Example

P(x) and P(y) :substitution = (x/y)

- P(x,x) and P(y,z) : (z/y)(y/x)
- P(x,f(y)) and P(Joe,z) : (Joe/x, f(y)/z)

- P(f(x)) and P(x) : can't do it!
- P(x) Ú Q(Jane) and P(Bill) ∨ Q(y): (Bill/x, Jane/y)

DIFFERENTIATE PREDICATE AND PROPOSITIONAL LOGIC.

Ans:

Sl.No	Predicate logic	Propositional logic
	Predicate logic is a generalization of propositional logic that allows us to express and infer arguments in infinite	A proposition is a declarative statement that's either TRUE or FALSE (but not both).
	Predicate logic (also called predicate calculus and first-order logic) is an extension of propositional logic to formulas involving terms and predicates. The full predicate logic is undecidable	Propositional logic is an axiomatization of Boolean logic. Propositional logic is decidable, for example by the method of truth table
	Predicate logic have variables	Propositional logic has variables. Parameters are all constant
	A predicate is a logical statement that depends on one or more variables (not necessarily Boolean variables)	Propositional logic deals solely with propositions and logical connectives
	Predicate logic there are objects, properties, functions (relations) are involved	Proposition logic is represented in terms of Boolean variables and logical connectives

	In predicate logic, we symbolize subject and predicate separately. Logicians often use lowercase letters to symbolize subjects (or objects) and uppercase letter to symbolize predicates.	In propositional logic, we use letters to symbolize entire propositions. Propositions are statements of the form "x is y" where x is a subject and y is a predicate.
	Predicate logic uses quantifiers such as universal quantifier ("∀"), the existential quantifier ("∃")	Prepositional logic has no quantifiers.
	Example Everything is green" as "∀x Green(x)" or "Something is blue" as "∃x Blue(x)".	**Example** Everything is green" as "G(x)" or "Something is blue" as "B(x)".

PROPOSITIONAL LOGIC

- Propositional Logic is concerned with statements to which the truth values, “true” and “false”, can be assigned. The purpose is to analyze these statements either individually or in a composite manner.
- A *proposition* is a declarative statement that’s either TRUE or FALSE (but not both).
- A proposition is a collection of declarative statements that has either a truth value "true” or a truth value "false". A propositional consists of propositional variables and connectives. We denote the propositional variables by capital letters (A, B, etc). The connectives connect the propositional variables.
    - Some examples of Propositions are given below −8
        - "Man is Mortal", it returns truth value “TRUE”
        - "12 + 9 = 3 – 2", it returns truth value “FALSE”
        - The following is not a Proposition −
        - "A is less than 2". It is because unless we give a specific value of A, we cannot say whether the statement is true or false.

Propositions	Not Propositions
3 + 2 = 32	Bring me coffee!
CS173 is Bryan’s favorite class.	CS173 is her favorite class.
Every cow has 4 legs.	3 + 2
There is other life in the universe.	Do you like Cake?

- **The symbols of the language:**
    - Propositional symbols (Prop): A, B, C,…
    - Connectives:

- ∧ and
- ∨ or
- ¬ not
- → implies
- ↔ equivalent to
- ⊗ xor (different than)
- ⊥, > False, True
- Parenthesis :(,).

Propositional Logic Semantics

Negation

- Truth tables define the semantics (=meaning) of the operators

Suppose *p* is a proposition.

The *negation* of *p* is written ¬*p* and has meaning:

"It is not the case that *p*."

- Ex. AI is NOT Bryan's favorite class.

P	¬*p*
T	F
F	T

Conjunction

Conjunction corresponds to English "and."

➢ p ∧ q is true exactly when p and q are both true.

➢ Ex. Amy is curious AND clever.

p	Q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Example

> Let p = “tired” and q = “hungry”
> Then p ∧ q = “tired and hungry”
> Also, ¬p ∧ q = “not tired and hungry”
> Or, p ∧ ¬q = “tired and not hungry”

Disjunction

Disjunction corresponds to English “or.”

- ➢ p ∨ q is when p or q (or both) are true.
- ➢ Ex. Michael is brave OR clever.

p	Q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Implication

Implication: p → q corresponds to English “if p then q,” or “p implies q.”

- ➢ If it is raining then it is cloudy.
- ➢ If there are 200 people in the room, then I am the Easter Bunny.

➢ If p then 2+2=4.

➢ If I am elected then I will lower the taxes

p	q	$p \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

Tautology

• Tautology is very similar to logical equivalence

• When all values are “true” that is a tautology

Example: p ≡ q if and only if p ↔ q is a tautology(BiConditional statement→If it rains ,we will get wet)--.XOR

Example: p ≡ ¬¬p is a tautology

Note all

Example: Show that p ^ (q V r) ≡ (p ^ q) V (p ^ r) ← statement

p	q	r	q V r	p ^ (q V r)	p ^ q	p ^ r	(p ^ q) V (p ^ r)	Statement
0	0	0	0	0	0	0	0	1
0	0	1	1	0	0	0	0	1
0	1	0	1	0	0	0	0	1
0	1	1	1	0	0	0	0	1
1	0	0	0	0	0	0	0	1
1	0	1	1	1	0	1	1	1
1	1	0	1	1	1	0	1	1
1	1	1	1	1	1	1	1	1

Propositional Logic: Truth Tables

Truth table for different connectives for Negation, Conjunction (AND), Disjunction (OR), Condition, Bicondition, NAND, NOR, XOR

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$p \uparrow q$	$p \downarrow q$	$p \oplus q$
T	T	F	T	T	T	T	F	F	F
T	F	F	F	T	F	F	T	F	T
F	T	T	F	T	T	F	T	F	T
F	F	T	F	F	T	T	T	T	F

Another way of representing truth table

p	q	p ^ q	¬(p ^ q)	¬p	¬q	¬p ^ ¬q	p V q	¬(p V q)	¬p ^ ¬q
0	0	0	1	1	1	1	0	1	1
0	1	0	1	1	0	1	1	0	0
1	0	0	1	0	1	1	1	0	0
1	1	1	0	0	0	0	1	0	0

Propositional Logic - special definitions

Contrapositives: $p \rightarrow q$ and $\neg q \rightarrow \neg p$

- Ex. “If it is noon, then I am hungry.”

“If I am not hungry, then it is not noon.”

Converses: $p \rightarrow q$ and $q \rightarrow p$

- Ex. “If it is noon, then I am hungry.”

“If I am hungry, then it is noon.”

Inverses: $p \rightarrow q$ and $\neg p \rightarrow \neg q$

- Ex. “If it is noon, then I am hungry.”

“If it is not noon, then I am not hungry.”

Propositional Logic: Logical Equivalences

- ***Identity***

$$p \wedge \mathrm{T} \equiv p$$

$$p \vee \mathrm{F} \equiv p$$

- ***Domination***

$$p \vee \mathrm{T} \equiv \mathrm{T}$$

$$p \wedge \mathrm{F} \equiv \mathrm{F}$$

- ***Idempotence***

$$p \vee p \equiv p$$

$$p \wedge p \equiv p$$

- ***Double negation***

$$\neg \neg p \equiv p$$

- ***Commutativity:***

$$p \vee q \equiv q \vee p$$

$$p \wedge q \equiv q \wedge p$$

- ***Associativity:***

$$(p \vee q) \vee r \equiv p \vee (q \vee r)$$

$$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$$

- ***Distributive*:**

$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$

- ***De Morgan's*:**

  $\neg(p \wedge q) \equiv \neg p \vee \neg q$ *(De Morgan's I)*

  $\neg(p \vee q) \equiv \neg p \wedge \neg q$ *(De Morgan's II)*

- ***Excluded Middle:***

  $p \vee \neg p \equiv \mathrm{T}$

- ***Uniqueness*:**

  $p \wedge \neg p \equiv \mathrm{F}$

- **A useful Logical Equivalence (LE) involving →:**

  $p \rightarrow q \equiv \neg p \vee q$

Clause- a special form

Literal

A single proposition and its negation. eg., p, $\neg p$

Clause

A clause is a disjunction of literals.

Eg., P∨Q∨¬R

Converting a compound proposition to the clausal form

Consider the sentence

¬(A→B) ∨(C→A)

1. Eliminate implication sign

   ¬(¬A∨B) ∨(¬C∨A)

2. Eliminate the double negation and reduce scope of "not sign" (De-Morgans Law)

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

3. Convert to conjunctive normal form (CNF)by using distributive and associative laws

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

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

4. Get the set of clauses

$(A \vee \neg C)$

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

Example Qns to Solve:

1.Obtain CNF of $(P \rightarrow Q) \wedge (Q \vee (P \wedge R)$.Find whether it is a tautology or not?

2.Obtain CNF of $(\neg P \rightarrow R) \wedge (Q \leftrightarrow P)$.

WKT $Q \leftrightarrow P = (Q \rightarrow P) \wedge (P \rightarrow Q)$

RESOLUTION

Resolution is a procedure used in proving that arguments which are expressible in predicate logic are correct. Resolution is a procedure that produces proofs by refutation or contradiction. Resolution leads to refute a theorem-proving technique for sentences in propositional logic and first order logic.

- Resolution is a rule of inference
- Resolution is a computerized theorem prover

Algorithm: Propositional Resolution

Let F be a set of axioms and P the theorem to prove.

1. Convert all the propositions of F to clause form.

2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.

3. Repeat until either a contradiction is found or no progress can be made

(a) Select two clauses. Call these the parent clauses.

(b) Resolve them together. The resulting clause called the resolvent, will be the disjunction of all of the literals of both of the parent clauses with the following exception: If there are any pair of literals L and ¬L such that one of the parent clauses contains L and the other contains ¬L, then select one such pair and eliminate both L and ¬L from the resolvent.

(c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

Example 1

1. If a triangle is equilateral then it is isosceles.

2. If a triangle is isosceles the two sides AB and AC are equal.

3. If AB and AC are equal then angle B and angle C are equal.

4. ABC is an Equilateral triangle.

Prove "Angle B is equal to angle C"

Propositional Logic

4. Equilateral(ABC)

1. Equilateral(ABC)→Isosceles(ABC)

2. Isosceles(ABC)→Equal(AB,AC)

3. Equal(AB,AC)→ Equal(B,C)

To prove " Equal (B,C)".

Convert to clausal form

1. ¬Equilateral(ABC) ∨Isosceles(ABC)

2. ¬Isosceles(ABC) ∨Equal(AB,AC)

3. ¬Equal(AB,AC) ∨ Equal(B,C)

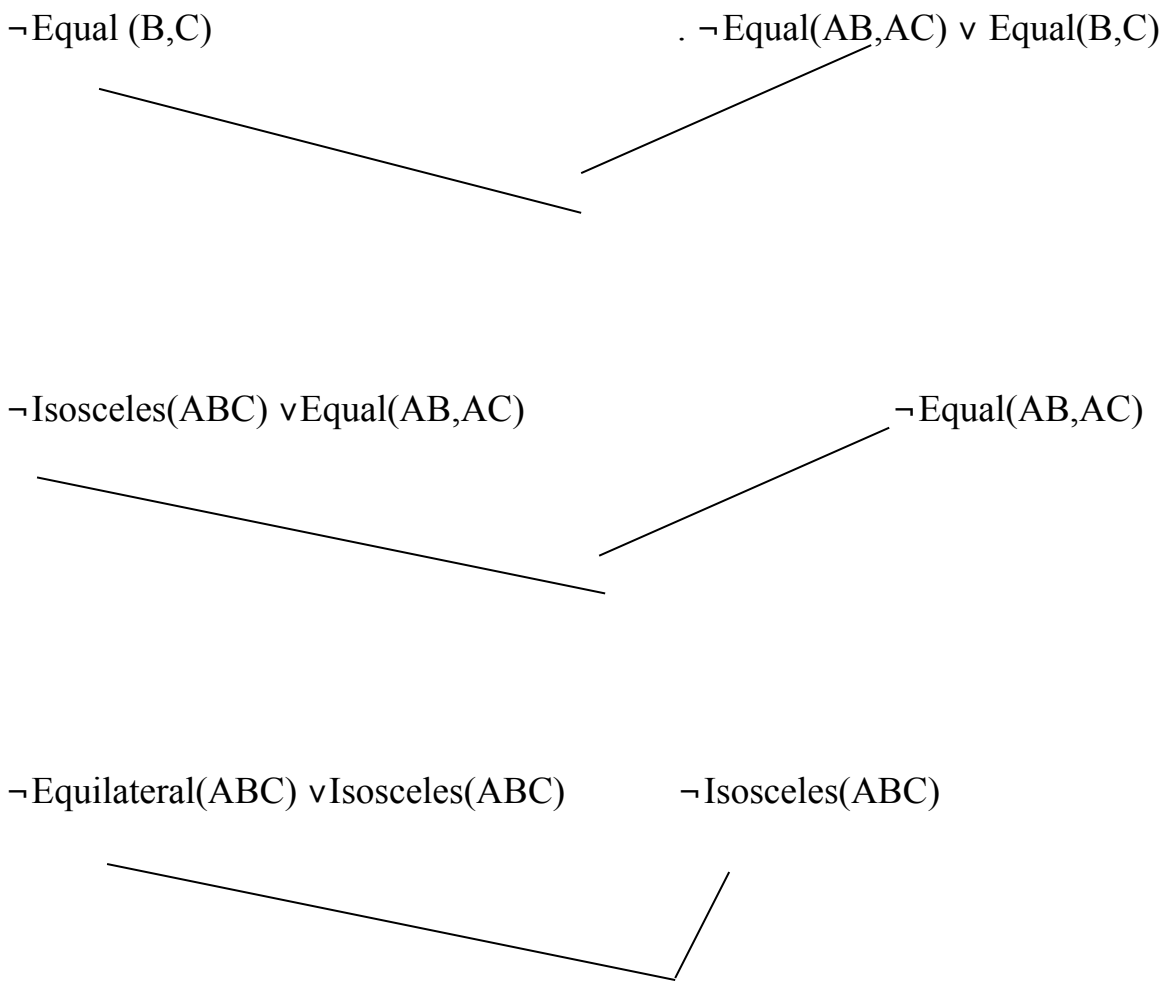
4. Equilateral(ABC)

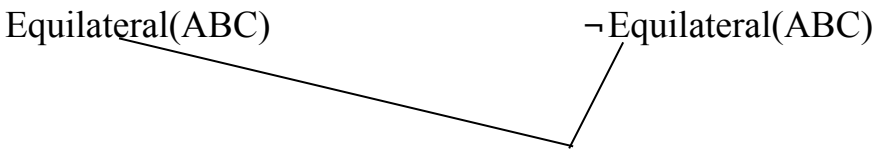
To prove ” Equal (B,C)”.

Let us disprove “Not equal B and C

¬Equal (B,C)”

Proof by contradiction





None

Thus we proved that angle B is equal to angle C

Example 2

1. Mammals drink milk.

2. Man is mortal.

3. Man is mammal.

4. Tom is a man.

Prove “Tom drinks milk” and Prove “Tom is mortal”

Propositional Logic

1. Mammal(Tom)→drink(Tom, Milk)

2. Man(Tom)→Mortal(Tom)

3. Man(Tom)→Mammal(Tom)

4. Man(Tom)

Convert to clausal form

1. ¬Mammal(Tom) ∨drink(Tom, Milk)

2. ¬Man(Tom) ∨Mortal(Tom)

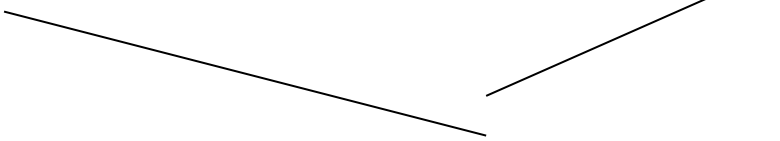
3. ¬Man(Tom) ∨Mammal(Tom)

4. Man(Tom)

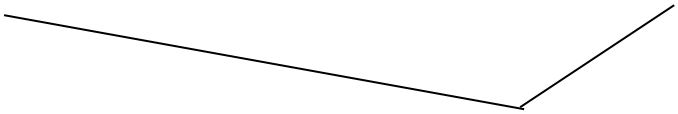
To prove ” Tom drinks milk”

Let us disprove "Not Tom drinks milk"➔"¬Drink (Tom, Milk)"

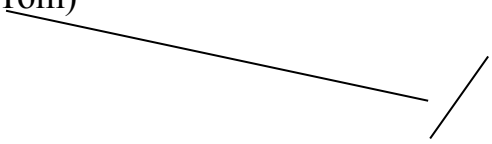
¬Mammal(Tom) ∨drink(Tom, Milk) . ¬Drink (Tom, Milk)



¬Man(Tom) ∨Mammal(Tom) ¬Mammal(Tom)



Man(Tom) ¬Man(Tom)



None

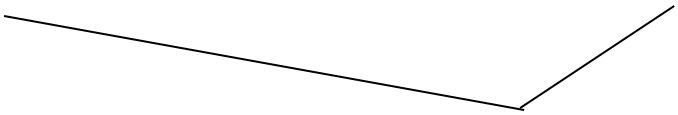
Thus we proved "Tom drinks Milk"

ii) To prove " Tom is Mortal"➔Mortal(Tom)

Let us disprove "Not Tom is Mortal" ➔¬Mortal(Tom)

¬Man(Tom) ∨Mortal(Tom) . ¬Mortal(Tom)

Man(Tom) ¬Man(Tom)



None

Thus we proved “Tom is Mortal”

Disadvantages of propositional logic

Propositional logic only deals with finite number of propositions.

KNOWLEDGE REPRESENTATION

In order to solve complex problems in artificial intelligence, we need large amount of knowledge and some mechanism for manipulating that knowledge to create solutions to new problems. Consider dealing with two different kinds of entities.

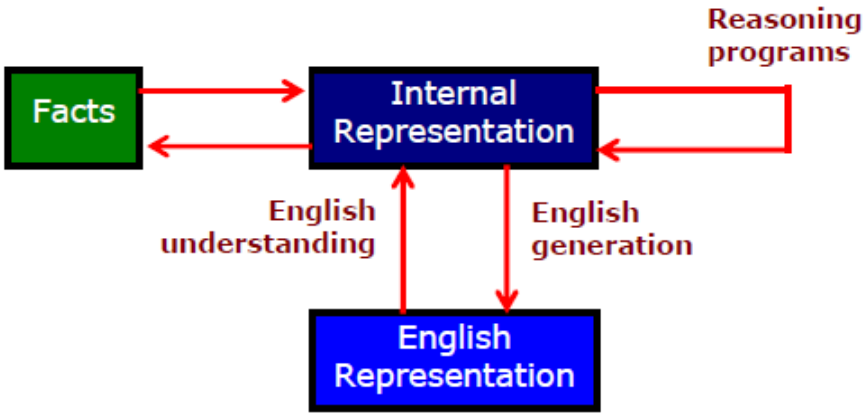
- Facts- truths in some relevant world. These are the things we want to represent.
- Representation- These are the things we will actually be able to manipulate.

One way to think of structuring these entities is as two levels.

- Knowledge level, at which the facts are represented.
- Symbol level, at which representations of objects at the knowledge level are defined in terms of symbols

There are two way mapping exists between facts and representation.

- Forward representation mapping – maps from facts to representation
- Backward representation mapping – maps from representation to facts



Mapping between facts and representation

English representation of these facts in order to facilitate getting information in to or out of the system.

Consider the English sentence

Spot is a dog

The fact represented by that English sentence can be represented in logic as

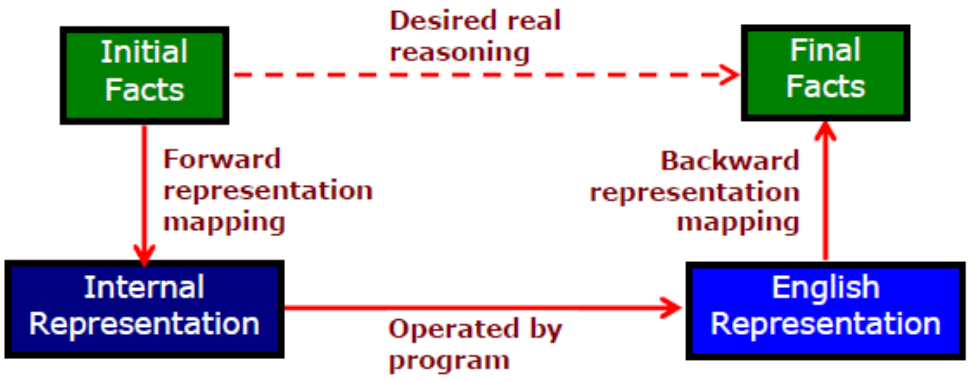
Dog(Spot)

Suppose we have logical representation of the fact that “All dogs have tail”

∀x:dog(x)→hastail(x)

Using appropriate backward mapping function, we could generate English sentence

Spot has a tail



Representation of facts (Expanded view)

APPROACHES TO KNOWLEDGE REPRESENTATION

A good system for the representation of structured knowledge in a particular domain should possess 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 to derive new structures corresponding to the new knowledge inferred from old..
(iii) **Inferential Efficiency**:- The ability to incorporate additional information into the knowledge structure that can be used to focus the attention of the inference mechanisms in the most promising directions.
(iv) **Acquisitional Efficiency** :- The ability to acquire new knowledge using automatic methods wherever possible rather than reliance on human intervention.

KNOWLEDGE REPRESENTATION SCHEMES

There are four types of knowledge representation

- Relational Knowledge
- Inheritable Knowledge
- Inferential Knowledge
- Procedural Knowledge

(i) Relational Knowledge

Relational knowledge provides a frame work to compare two objects based on equivalent attributes. This knowledge associates element of one domain with another domain. Relational knowledge is made up of objects consisting of attributes as their corresponding associated values. The results of this knowledge type is mapping of elements among different domains.

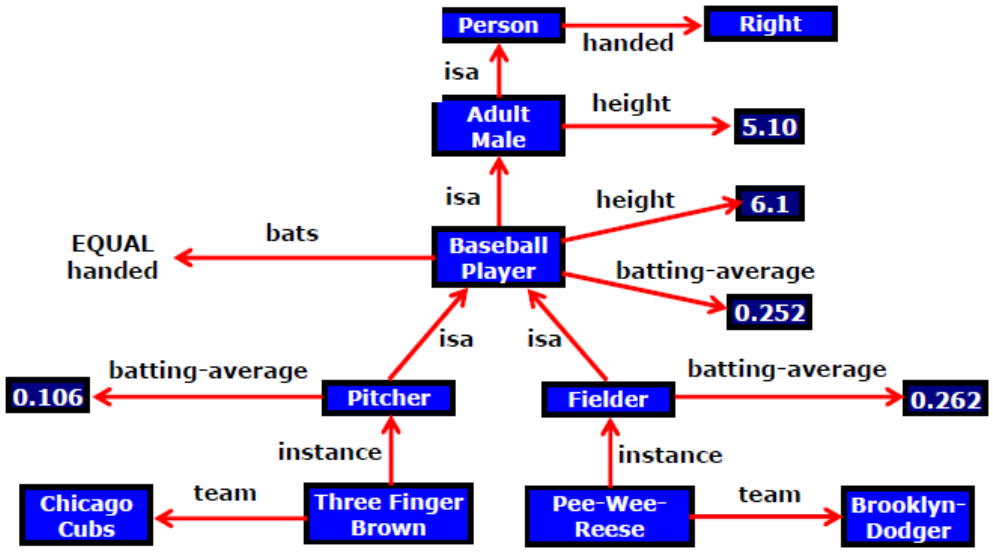
The table below shows a simple way to store facts. The facts about the set of objects are put systematically in columns.

Player	Height	Weight	Bats - Throws
Aaron	6-0	180	Right - Right
Mays	5-10	170	Right - Right
Ruth	6-2	215	Left - Left
Williams	6-3	205	Left - Right

Given the fact that it is not [possible to answer a simple question such as "Who is the heaviest player?". But, if the procedure for finding the heaviest player is provided, then these facts will be enable that procedure to compute an answer. This representation provides only little opportunity for inference.

(ii) Inheritable Knowledge

Inheritable knowledge is obtained from associated objects. It describes a structure in which new objects are created which may inherit all or subset of attributes from existing objects.



Inheritable knowledge representation

The directed arrow represent attributes originates at object being described and terminates at object or its value. The box nodes represent objects and values of the attributes.

Viewing node as a frame: Baseball Player

isa :Adult-Male

bats : Equal to handed

height : 6.1

bating average:0.252

Algorithm: Property of Inheritance

The steps for retrieving a value v for an attribute A of an instance object O

i) Find object O in the knowledge base.

ii) If there is a value for the attribute A, then report that value.

iii) Otherwise, see if there is a value for the attribute instance. If not then fail.

iv) Otherwise, move to the node corresponding to that value and look for a value for the attribute A. If one is found, report it.

v) Otherwise, do until there is no value for the isa attribute or until an answer is found.

(a) Get the value of "isa" attribute and move to that node.

(b) See if there is a value for the attribute A; If yes, report it.

Team(Pee-Wee-Reese)=Brooklyn Dodger

Batting average(Three finger Brown)=0.106

Height(Pee-Wee-Reese)=6.1

Bats(Three finger Brown)=right

iii) Inferential Knowledge

Inferential knowledge is inferred from objects through relations among objects. A word alone is a simple syntax, but with the help of other words in phrase the reader may infer more from a word; this inference with in linguistic is called semantics.

This knowledge generates new information from the given information. This information does not require further data gathering from source, but does require analysis of the given information to generate a new knowledge.

Example

Given a set of values and relations, one may infer other values or relations. Predicate logic is used to infer from a set of attributes. Inference from a predicate logic uses a set of logical operations to relate individual data.

i) Wonder is a name of a dog.

Dog(Wonder)

ii) All dogs belong to the class of animals.

$\forall x: dog(x) \rightarrow animal(x)$

iii) All animals either live on land or in water.

$\forall x: animal(x) \rightarrow live(x, land) \vee live(x, water)$

From these three statements we infer that

Wonder lives either on land or on water

iv) Procedural Knowledge

Procedural Knowledge specifies what to do when. The procedural knowledge is represented in program in many ways. The machine uses the knowledge when it executes the code to perform a task.

Example: A parser in a natural language has the knowledge that a noun phrase may contain articles adjectives and nouns. Thus accordingly call routines that know how to process articles , adjectives and nouns.

ISSUES IN KNOWLEDGE REPRESENTATION

The fundamental goal of knowledge representation is to facilitate inference from knowledge. The issues that arise while using knowledge representation techniques are many. Some of these are explained below.

- Any attributes of objects so basic that they appear in almost every problem domain?. If there are we need to make sure that they are handled appropriately in each of the mechanisms we propose.
- Any important relationship that exists among object attributes?
- At what level of detail should the knowledge be represented?
- How sets of objects be represented?
- Given a large amount of knowledge stored in a database, how can relevant parts be accessed?

i) Important Attributes

There are two attributes that are of very general significance

- Isa
- Instance

These attributes are important because they support property inheritance. They represent class membership and class inclusion and the class inclusion is transitive

ii) Relationships among Attributes

The attributes that are used to describe objects are themselves entities. The relationship between the Each entity have four properties independent of the specific knowledge they encode. They are:

(a) Inverses

(b) Existence in an isa hierarchy

(c) Techniques for reasoning about values

(d) Single-valued attributes

(a) Inverses:

Entities in the world are related to each other in many different ways. Relationships are attributes such as **isa, instance and team.** There are two good ways to represent the relationship.

The first is to represent both relationships in a single representation

team(Pee-Wee-Reese, Brooklyn-Dodgers) can equally easily be represent as a statement about Pee-Wee-Reese or about the Brooklyn-Dodgers.

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

team = Brooklyn-Dodgers
team-members = Pee-Wee-Reese;…

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

(b) Existence in an isa hierarchy

The attribute height is the specialization of general attribute physical size which is in turn a specialization of physical attribute. The generalization specialization attributes are important because they support inheritance.

(c) Techniques for reasoning about values

This is about reasoning values of attributes not given explicitly. Several kind of information are used in reasoning like

- Information about the type of value. Eg height must be a unit of length.
- Constraints on the value. Eg age of a person cannot be greater than the age of persons parents.
- Rules for computing the value when it is needed. Eg bats attribute. These rules are called backward rules. Such rules are also called if-needed rules.

- Rules that describe actions that should be taken if a value ever becomes known. These rules are called forward rules or sometimes if-added rules.

(d) Single-valued attributes

This is about specific attribute that is guaranteed to take unique value. Eg. A baseball player can at a time have only a single height and be a member of only one team.

(iii) Choosing the granularity of representations

It deals with what level should the knowledge be represented. The primitives are

- Should there be a small number or should there be a large number of low level primitives or high level facts
- High level facts may not be adequate for inference while low level primitives may require a lot of storage.

Example, Consider the following fact

John spotted Smith

This could be represented as

Spotted(John, Smith)

Or

Spotted(agent(John), object(smith))

Such representation would make it easy to answer question such as

Who spotted Smith?

Suppose we want to know

Did John see smith?

Given only one fact, but we cannot discover the answer. So we can add another fact

Spotted(x, y)→saw(x, y)

We can now infer the answer to the question

(iv)Representing set of objects

Certain properties of objects that is true as member of a set but not as individual. Consider the assertion made in the sentences

"There are more sheep than people in Australia" and English speakers can be found all over the world".

To describe these facts, the only way to attach assertion to the sets representing people, sheep and English. The reason to represent sets of object is: If a property is true for all or most elements of a set, then it is more efficient to associate it once with the set rather than to associate explicitly with every elements of the set.This is done in different ways:

- In logical representation through the use of universal quantifier,
- In hierarchical structure where node represents sets, the inheritance propagate set level assertion down to individual.

Example: assert large (elephant);

Remember to make clear distinction between,

- Whether we are asserting some property of the set itself, means, the set of elephants is large.
- Asserting some property that holds for individual elements of the set, means, anything that is an elephant is large.

There are three ways in which sets may be represented

- Names
- Extensional definition
- Intentional definition

(v) Finding the right structures as needed

To access the right structure for describing a particular situation, it is necessary to solve all the following problems,

- How to perform an initial selection of the most appropriate structure
- How to fill in appropriate details from the current situation
- How to find a better structure if the one chosen initially turns out not to be appropriate.
- What to do if none of the available structure is appropriate
- When to create and remember a new structure
    - Selecting an Initial structure
        - ➢ Index the structure
        - ➢ Pointer to all the structures
        - ➢ Locate one major due in the problem description
    - Revising the choice when necessary

Proof by Resolution: Example 1

If either C173 or C220 is required, then all students will take computer science. C173 and C240 are required. Prove that all students will take computer science.

We formalize the proof as follows:

```
P1. (C173 OR C220) -> ACS
P2. C173
P3. C240
Prove: ACS
```

We then rewrite our hypotheses in conjunctive normal form:

```
P1: (NOT C173 OR ACS) (NOT C220 OR ACS)
P2: C173
P3: C240
```

Then we use proof by contradiction, by asserting the clauses of the premises and the negation of the conclusion, and deriving false.

```
1. NOT C173 OR ACS    Premise
2. NOT C220 OR ACS    Premise
3. C173               Premise
4. C240               Premise
5. NOT ACS            Negation of conclusion
6. NOT C173           L1, L5, resolution
7. FALSE              L3, L6, resolution
```

Proof by Resolution: Example 2

Either Heather attended the meeting or Heather was not invited. If the boss wanted Heather at the meeting, then she was invited. Heather did not attend the meeting. If the boss did not want Heather there, and the boss did not

invite her there, then she is going to be fired. Prove Heather is going to be fired.

1.	A OR NOT I	Premise
2.	NOT W OR I	Premise
3.	NOT A	Premise
4.	W OR I OR F	Premise
5.	NOT F	Negation of conclusion
6.	W OR I	L4, L5, resolution
7.	I	L2, L6, resolution, idempotence
8.	A	L1, L7, resolution
9.	FALSE	L3, L8, resolution

Proof by Resolution: Example 3

Either taxes are increased or if expenditures rise then the debt ceiling is raised. If taxes are increased, then the cost of collecting taxes rises. If a rise in expenditures implies that the government borrows more money, then if the debt ceiling is raised, then interest rates increase. If taxes are not increased and the cost of collecting taxes does not increase then if the debt ceiling is raised, then the government borrows more money. The cost of collecting taxes does not increase. Either interest rates do not increase or the government does not borrow more money.

Prove either the debt ceiling isn't raised or expenditures don't rise.

1.	T OR NOT E OR D	Premise
2.	NOT T OR C	Premise
3.	(E AND NOT G) OR NOT D OR I	Premise
4.	T OR C OR NOT D OR G	Premise
5.	NOT C	Premise
6.	NOT I OR NOT G	Premise

7. D AND E	Negation of conclusion
8. (E AND NOT G) OR I	L3, L7, resolution
9. C OR NOT D OR G	L2, L4, resolution
10. C OR G	L7, L9, resolution
11. G	L5, L10, resolution
12. NOT I	L6, L11, resolution
13. E AND NOT G	L8, L12, resolution
14. NOT G	L13, tautology
15. FALSE	L11, L14, contradiction