Knowledge Representation.

We have seen many general methods for manipulating knowledge using Search.

Different ways of representing the knowledge :

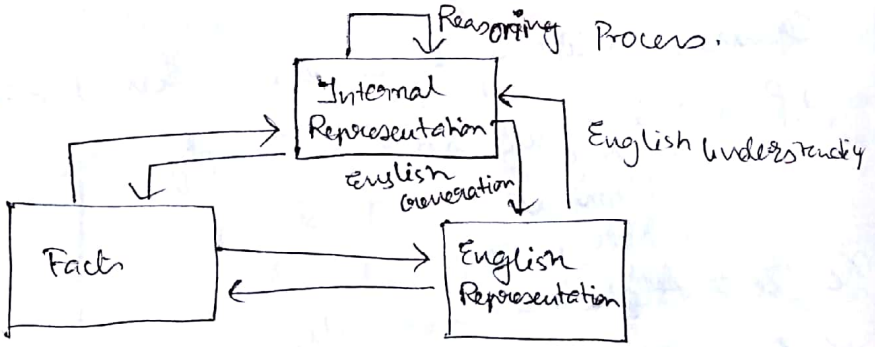
The entities pertaining to any knowledge representation

I Facts : Truth in some real world.
(These are things we want to represent).

II Representation of Truth in some chosen formalism :
These are things we actually are able to manipulate.

Each knowledge representation must associate with some mapping function :

- Function that map from facts to chosen representation
- Function that map from representation to facts.



Knowledge Representation using Mathematical Knowledge :

① All dogs have tails : $\forall x, dog(x) \rightarrow hastail(x)$.

② Spot is a dog : $dog(spot)$

③ Spot has a tail : $hastail(spot)$.

It is important to note that the mapping function are not one to one.

All dogs have tails → Every dog has atleast one tail.
→ Every dog has several tails.

Every dog has a tail → Every dog has atleast one tail.
→ There is tail that every dog has.

Both represent the same fact that every dog has atleast one tail.

Propositional Logic

Knowledge representation using propositional logic is appealing for two reasons —

- It's simple to deal with
- There exists a decision procedure for it

English Sentence	Propositional Logic
It is raining	RAINY
It is sunny	SUNNY
It is windy	WINDY

If it is not raining then it is sunny

~RAINY → SUNNY

Limitations of Propositional Logic

I
- Crayon is a Student → Crayon Student
- Ritwaj is a Student → Ritwaj Student

There are two totally seperate assertion. It is not possible to draw any conclusion about similarities b/w Crayon & Ritwaj

II All men are mortal.

Predicate Logic

i) Marcus was a man. —— Man(Marcus)

(ii) Marcus was a pompeian. —— Pompeian(Marcus)

(iii) All Pompeians were Romans —— $\forall x$ Pompeian(x) → Roman(x)

(iv) Ceaser was a Ruler. —— Ruler(Ceaser)

~~(v) All Romans were loyal to Ceaser~~

(v) All Romans were either loyal to Ceaser or hated him. —— $\forall x$ Roman(x) → loyalto(x, ceaser) ∨ hate(x, ceaser).

(Existential Q.)

~(loyalto(ceeser) ∧ ~hate(ceaser))

(vi) Everyone is loyal to someone $\forall x \exists y$ s.t. loyalto(x, y)

(vii) People only try to assassinate ruler they are not loyal to.

~~$\forall x \forall y$ people(x) ∧ ruler(y) ∧ tryassassinate(x, y)~~

$\forall x \forall y$ people(x) ∧ ruler(y) ∧ tryassassinate(x, y).

(ix) Marcus tried to assassinate ~(loyalto(x, y)) ceaser

tryassassinate(Marcus, Ceaser).

Now was Marcus loyal to Ceaser?

~loyalto(Marcus, Ceaser).

(7)

Person(Marcus) ∧ Ruler(Ceaser) ∧ tryassassinate(Marcus, Ceaser)

(1)

Person(Marcus) ∧ tryassassinate(Marcus, Ceaser)

(8)

Person(Marcus) (9) → Man(Marcus) (1) → No unspecified goal remain

Hence, Marcus was not loyal to Ceaser.

In order to prove the goal, we need to use rules of inference to transfer one goal into another goal that in turn to be transformed and so on until no unsatisfied goal remains.

Computational Predicates

When the number of facts are not very large and if the facts themselves are constructured then we can express them as combination of individual predicates.

Suppose we want to express simple facts as

gt(1,0) lt(0,1)
gt(2,1) lt(1,2)
⋮ ⋮

} Infinitely very large no. of such facts

It is extremely inefficient to store these

So instead of storing these predicates, we may compute each one as we need them.

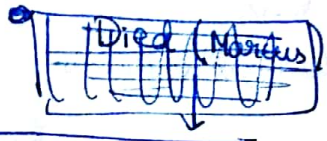
So we invoke a procedure instead of searching database.

1. Marcus was a man
2. Marcus was a Pompeian.
3. Marcus was born in 40 AD
4. All men are mortal.
5. All pompeians died in 79 AD when volcano erupted.
6. No mortal lives beyond 150 yrs.
7. It is now 2017 AD.

Q Is Marcus Alive

Sol.
1. Man(Marcus)
2. Pompeian(Marcus)
3. Born(Marcus, 40 AD)
4. ~~Mortal(Marcus)~~ $\forall x$, man(x) → mortal(x)
5. $\forall x$ erupted(Volcano, 79AD), Pompeian(x) → died(x, 79AD)
6. ~~$\forall x$, died(x, 150yrs)~~
6. ~~$\forall x$, (age(x) ≥ 150) → dead(x)~~
6. $\forall x$, (mortal(x) $\wedge$ age(x) ≥ 150) → dead(x)
7. now = 2017.

Is Marcus Alive?



~alive (Marcus, Now)

↓

dead (marcus)

↓ ⑥

Mortal (Marcus) ∧ age(Marcus) ≥ 150

↓ ④, ⑦ & ③

Man(Marcus) ∧ |Born(Marcus) − Now|

↓

Specified goal remains.

~alive (Marcus, Now)

↓

dead (Mar

⑦ Sir notes

~alive (Marcus, now)

↓

dead (Marcus)

↓

died(Marcus, t_1)

∧ gt(now, t_1)

↓

Pompeian (Marcus)

∧ gt (Now, 79)

↓

gt (now, 79)

↓

gt(2014, 79)

↓

nill.

Resolution

$E_1 \vee E_2$ ⟶ Disjunction

Disjuncts

$E_3 \wedge E_4$ ⟶ Conjunction.

Conjuncts.

Predicate : Predicates are functions that map object arguements into True or False values.

$\forall$ {forall, forevery} ⇒ Universal Quantifier.

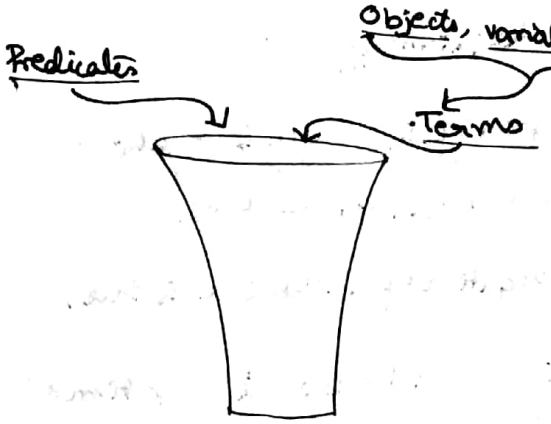
$\exists$ {There exists} ⇒ Existential Quantifier

Terms

1. Domain's objects are terms.
2. Variables ranging over domain's object are called terms.
3. Functions are terms.

Objects

1. Arguements to a function are objects
2. The value returned by a function are objects.



Suppose Object = A
variable = x
Predicate = On.
So output = On(A, x)

Negation.

Negation of the atomic formula is ~On(A,x)

Atomic formula & Negation of Atomic form

together called as Literals

Now, Connectives ($\vee, \wedge, \rightarrow, \sim$), Literals, Quantifiers ($\exists, \forall$)

Well formed Formulas.

Modus Ponens: If there is an axiom of the form $E_1 \rightarrow E_2$, & there is another axiom E_1, then E_2 logically follows.

Modus Tolens: If there is an axiom of the form $E_1 \rightarrow E_2$, and there is another axiom $\sim E_2$, then $\sim E_1$ logically follows.

Resolution: If there is an axiom of the form $E_1 \vee E_2$, and there is another axiom $\sim E_2 \vee E_3$, then $E_1 \vee E_3$ logically follows.

Resolvement of $E_1 \vee E_2$ & $\sim E_2 \vee E_3$

Resolution Proof: [Proof by Refutation].

The Resolution theorem proving strategy is to show that the negation of a theorem cannot be true.

Step 1: Assume that the negation of a theorem is true.

Step 2: Show that the axioms and assumed negation of the theorem together determine something to be true, that cannot be true.

Step 3: Conclude that the assumed theorem cannot be true since it leads to contradiction.

Step 4: Conclude that the theorem is true since the assumed negation of the theorem can not be true.

Proof by Refutation involves proving the theorem by showing that its negation can not be true.

The Resolution procedure operates by taking two clauses that contain the same literals in both clauses. The literals must occur in their positive form in one clause and negative form in another clause.

The resolvement is obtained by combining all the literals from both the parent clauses except the ones that cancel.

If the clause that is produced is an empty clause, then contradiction has been formed.

If there is no contradiction exists then the resolution procedure will never end.

Well formed Formula → (convert) Clause Forms.

Converting well-formed formula into Clause Form.

They are all the same. ‖ Clause Form ⟷ Canonical Form ⟷ Conjunctive Normal Form.

Step1 Eliminate '→' symbol, using $a \rightarrow b \cong \sim a \vee b$

Step2 Reduce the scope of '~':

(i) $\sim(\sim a) = a$

(ii) Using De-Morgan's Law: $\sim(a \vee b) = \sim a \wedge \sim b$

$\sim(a \wedge b) = \sim a \vee \sim b$

(iii) Standard Correspondence b/w Quantifiers: $\sim \forall x\, P(x) = \exists x \sim P(x)$

$\sim \exists x\, P(x) = \forall x \sim P(x)$.

Step3 Standardise the variable:

Each quantifier binds to a unique variable

$\forall x\, P(x) \vee \forall x\, Q(x)$

↓ To be written as

$\forall x\, P(x) \vee \forall y\, Q(y)$

Step 4 Move all the quantifiers to the left side of the formula

Prefix of quantifiers followed by the Matrix —

[Quantifier] [Matrix (Literals.)]

After following the 4th Step, the formula is in
② Prenex Normal Form

Step 5 Eliminate the existential quantifiers.

Using Skolem function Skolem Constant.

$\exists y$ President(y) ⟶ President(S_1)
(List of presidents only) ↳ Skolem Co

S_1 is a function without any argument that somehow produce a value that satisfies the predicate President

Problem in removing existential quantifier.

Suppose the existential quantifier occurs within the s of universal quantifier

$\forall x \exists y$ Father(y, x).

In this case, we use Skolem function.

$\forall x$ Father($S_2(x)$, x) // Function can be to a predicate
↳ Skolem Function.

Step 6 Drop the prefix (of quantifiers)

[X] [Matrices]

Step 7 Convert the matrix into conjunction of disjuncts, using associative and distributive properties.

$(\_ \vee \_) \wedge (\_ \vee \_ \vee \_) \wedge (\_ \vee \_)$

Step 8 Call each conjunct as a seperate clause.

Step 9 Standardize apart the variables in the set of clause generated.

Resolution in Propositional Logic.

Example.

Given, P, $(P \wedge Q) \rightarrow R$, $(S \vee T) \rightarrow Q$, t, Prove r.

Convert the whole to clause.

clause form:

① P

$\sim(P \wedge Q) \vee R$

$\equiv \sim(P \wedge Q) \vee R$

② $\sim P \vee \sim Q \vee R$

$\sim(S \vee T) \vee Q = (\sim S \wedge \sim T) \vee Q$

③ $= (\sim S \vee Q) \wedge (\sim T \vee Q)$

③' $\sim S \vee Q$

④ $\sim T \vee Q$

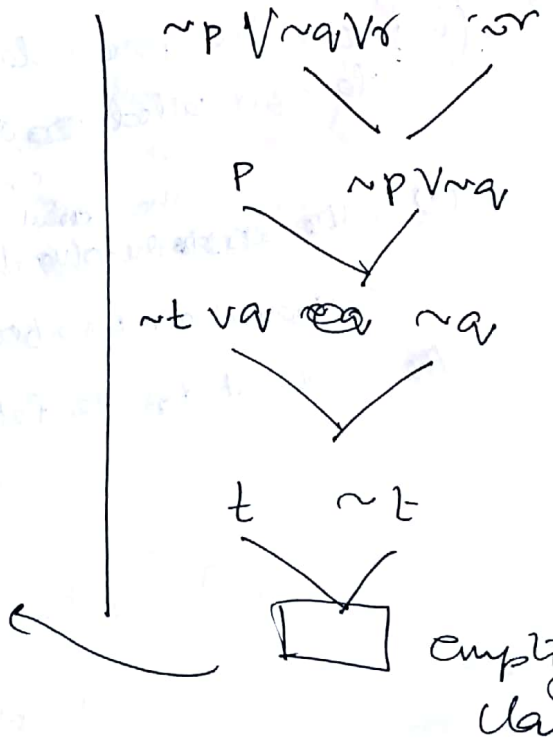
⑤ t

⑥ $\sim R$

$\sim R$ cannot be r

r is true

Hence proved.



13/11

Resolution in Propositional Logic.

Procedure for producing a proof by resolution of proposition with respect to set of axioms 'F'.

Step 1 Convert all axiom propositions of 'F' into clause form.

Step 2 Negate 's' and convert the result into clause form. Add it to the set of clauses obtained in step 1.

Step 3 Repeat the following until either a contradiction is found No progress can be made.

(a) Select two clauses and call them as the parent cl

(b) Resolve the parent clause and the resulting clause is called resolvent.

(c) If the resolvent is an empty clause then conjunction of all the literals from the parent clauses, except the literals that cancel contradiction has been found, otherwise add the resolvent to the set of clauses available for the procedure.

Unification Algorithm.

In propositional logic, it is easy to find two literals that can be true at the same time.

But it is not easy in predicate logic.

example:

Student(chayan) ~Student(chayan) → It is a contradiction

Student(chayan) ~Student(Spot) → It is not a contradiction.

So we need a matching process, that compares two literals and discover whether there exists a set of substitution that makes them identical.

- Matching Rules

Different Predicate can not match with each other

Different constant can not match with each other.

Only identical constant/predicates can match with each other.

A variable can match with another variable/ any constant / another predicate expression.

(*) when matching with another predicate expression, the matching predicate expression should not contain any instance of the variable which is being matched.

example

P(a,x) P(y,z) → y/x → P(y,x) P(y,z) → z/x → P(y,z) P(y,z)

This example also shows a complication, which is that the variable x is used twice, once for y & then z. This is not preferable.

This problem can be overcomed by —

P(x,x) P(y,z) → y/x → P(y,y) P(y,z) → z/y → P(y,z) P(y,z)

Substituting all remaining instances of x with y.

hate (x, y)
hate (Marcus, y)
hate (Marcus, Ceaser)
hate (Marcus, Rule (Rome))] → Argument can be another expression.

Objective of unification Algorithm.

To find atleast one set of substitution that cause two literals to match

example

hate(x, y) ~ (Marcus/x y/z)
hate (Marcus, z) ~ (Marcus/x z/y)

Algorithm_unify (L_1, L_2)
{

Step 1: If, L_1 & L_2 are both variable or constant then —
(a) If L_1 & L_2 are identical then return NIL.
(b) Else if L_1 is a variable and if L_1 occurs in L_2, then return (FAIL) else return L_2/L_1
(c) Else if L_2 is a variable and L_2 occur in L_1 then return (FAIL) else return L_1/L_2.
Else, return (FAIL).

Step 2: If the initial symbols in L_1 and L_2 are not identical then return (FAIL) // If predicate name differs

Step 3: If L_1 & L_2 have different number of arguments, then return (FAIL).

Step 4: Set SUBST to NIL.
set containing list of substitution at end.

Step 5: For i ← 1 to number of arguements in L_1 or L_2,

(a) Call Unify with i^{th} arguement of L_1 & i^{th} arguement of L_2, and return the result in S.

(b) If S contains FAIL, then return (FAIL).

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

(i) Apply S to remainder of both L_1 & L_2.

(ii) SUBST ← Append (S, SUBST)

Step 6: Return SUBST.

14/11 1) John likes all kinds of food.

2) Apples are food.

3) Chicken is food.

4) Anything anyone eats and is not killed by is food.

5) Bill eats peanuts and is still alive

6) Sue eats everything Bill eats.

- Translate these sentences into formulas in predicate logic.
- Convert the formula into clause form.
- Prove that John likes peanuts.

Sol. (1) $\forall x$, Food $(x) \rightarrow$ Like (John, x)

(2) food (apple)

(3) Food (Chicken

(4) $\forall x \forall y$ Eats $(y, x) \wedge \neg$ killed $(y) \rightarrow$ food (x).

(5) Eats (Bill, Peanuts) $\wedge$ Alive (Bill).

(6) $\forall x$ Eats (Bill, x) $\rightarrow$ Eats (Sue, x).

Clause form :

1. $\sim food(x) \vee likes(John, x)$
2. $food(apple)$
3. $food(chicken)$
4. $\sim eat(x, y) \vee killed(x) \vee food(x)$
5. $eats(Bill, peanuts)$
6. $alive(Bill)$.

$\sim eats(Bill, x) \vee eats(Sue, x)$.

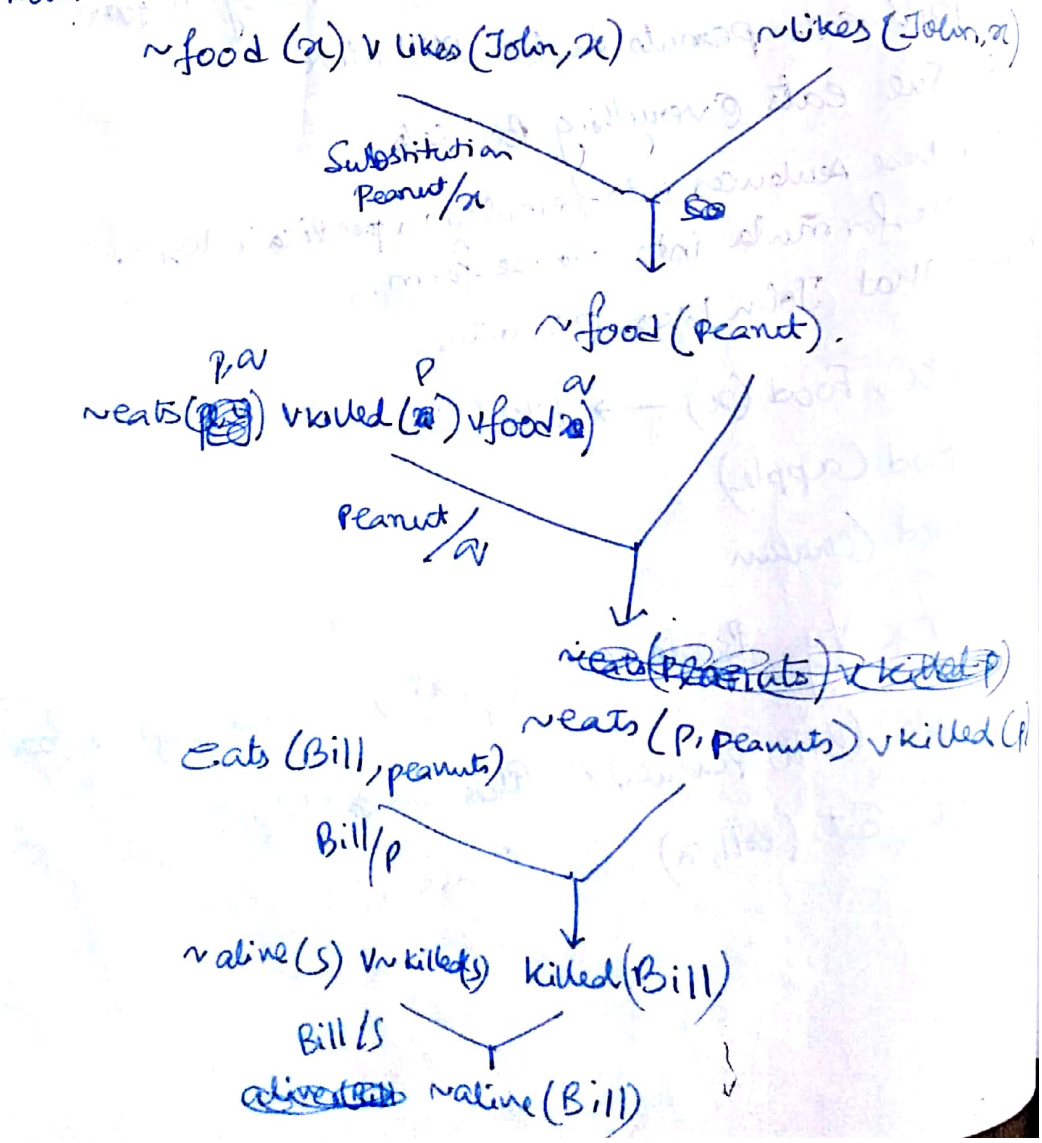
7. Alive means not killed
   $\forall x\ alive(x) \rightarrow \sim killed(x)$
   $\sim alive(x) \vee \sim killed(x)$

To prove $likes(John, Peanuts)$.

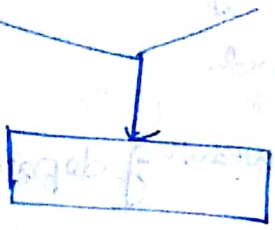
Now, proof by Refutation [If cannot prove contradiction, then theorem is true]

So, $\sim likes(John, Peanuts)$.

Now,



alive (Bill) ~alive (Bill)



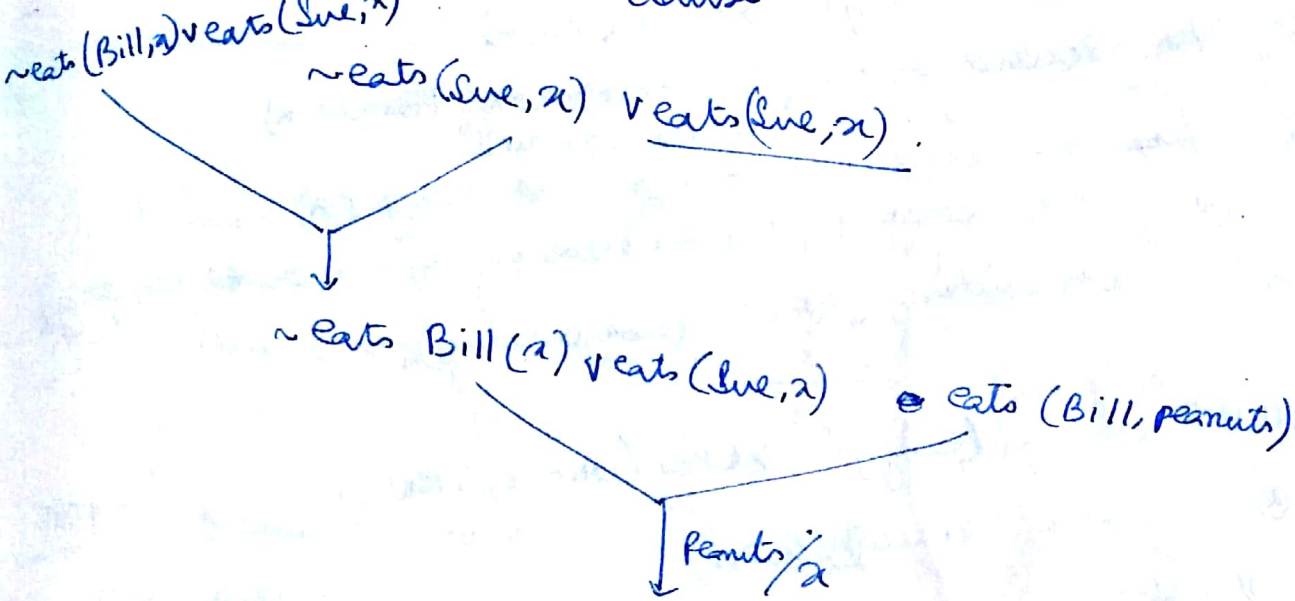
empty clause.

Since full of contradictions, so ~Likes (John, peanuts) is false.

So, Likes (John, Peanuts) is true.

- Using Resolution to answer question "what food does Sue eat?"

Lets take a dummy clause. eats (Sue, x).



So ∀ x = peanuts,

So Sue eats peanuts.

(1) Steve only likes easy courses.

(2) Science courses are hard.

(3) All courses in basketweaving department are easy.

(4) BK 301 is basketweaving course.

[redacted] :

Use resolution to answer "what course would Steve like?"

Sol:

(1) $\forall x$, course (x, easy) $\rightarrow$ likes (Steve, x).

(2) course (x, science) $\rightarrow$ hard (x).

(1) course (easy) $\rightarrow$ likes (course).

(2) course (science, hard).

(3) $\forall x$, course (x, basketweaving) $\rightarrow$ easy (x).

(3) $\forall x$, dept (basketweaving, x) $\rightarrow$ course (x, easy)

(4) dept (BK301, basketweaving).

(1) $\forall x$ course (x, easy) $\rightarrow$ likes (course, Steve)

(2) course (science, hard)

(3) $\forall x$ dept (basketweaving, x) $\rightarrow$ course (x, easy)

(4) dept (BK301, basketweaving).

Clause form

(1) $\sim$ course (x, easy) $\vee$ likes (Steve, x)

(2) course (science, hard)

(3) $\sim$ dept (basketweaving, x) $\vee$ course (x, easy)

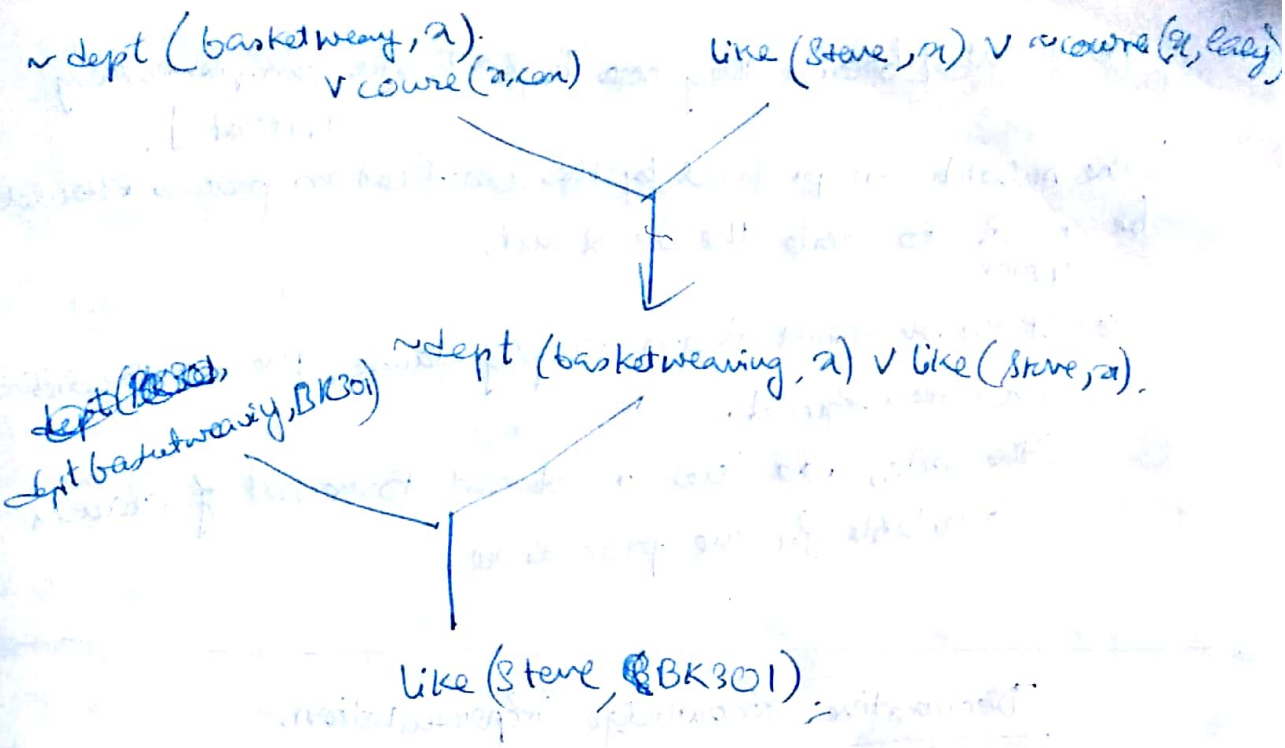
(4) dept (BK301, basketweaving).

$\sim$likes (Steve, x) likes (x, Steve). $\sim$course (x, easy)

$\sim$likes (Steve, x) $\vee$ like (Steve, x)

$\sim$course (x, easy) $\vee$ likes (

$\sim$like (Steve, x) $\vee$ $\sim$course (x, easy)



15/11

Resolution in Predicate Logic.

'F' → Given set of statements. Prove 'S'.

Step 1 Convert all the statements of 'F' to predicate logic and then to clause form.

Step 2 Negate 'S' and convert the result into clause form. Add it to the set of clauses obtained in the (previous) step 1.

Step 3 Repeat until either contradiction is found or no progress is made or a predetermined amount of effort has been expended.

Different from resolution in propositional Logic.

(a) Select two clauses & call them as parent clause.

(b) Resolve the parent clauses.
The resolvant will be the disjunction of all the literals from both the parents clauses with appropriate substitutions, with the following exceptions:

If there are two literals, T_1 & $\sim T_2$, in different parent clauses, and if they are unifiable, then both T_1 & $\sim T_2$ should not appear in the

resolvent [because they T_1 & $\sim T_2$ are complementary literals].

The substitution produced by the unification procedure should be used/applied to create the resolvant.

(c) If the resolvant is an empty clause, then contradiction has been found.
Otherwise, add the resolvent to the set of clauses available for the procedure.

Declarative Knowledge Representation.

The knowledge representation using predicate logic and propositional logic is often useful in representation of simple facts.

The main advantage is these can be combined with powerful inference mechanism called Resolution.

Major Disadvantage: It is very difficult to represent complex knowledge using these representations.

Properties of a Good Knowledge Representation. ©[Imp]

1. Representational Adequacy: The ability to represent all kinds of knowledge that are needed in the particular domain.
2. Inferential Adequacy: It is the ability to manipulate the representative structure in such a way that to derive a new struct corresponds to new knowledge.
3. Inferential Efficiency: The ability to incorporate into the knowledge structure, the additional information that can be used to focus on the attention of the inference mechanism in a most promising direction.

4. Aquisitional Efficiency : The ability to acquire new knowledge / information easily.

Several techniques has been proposed to meet their properties and they are classified into ——

Declarative Method vs Procedural Method

Each fact is stored only once, regardless of number of different ways it can be used.

It is easy to add the new facts to the system without changing the facts or small problems.

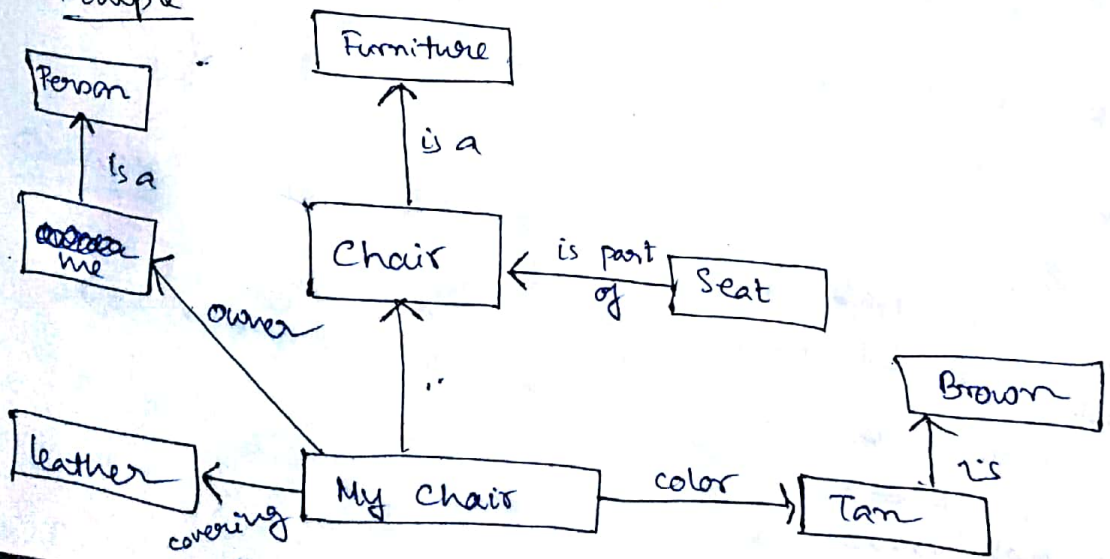
1. It is easy to represent the knowledge of how to do things.
2. It is easy to represent the facts that does not fit into many simple declarative schemes.
3. It is easy to incorporate heuristic knowledge of how to do things efficiently.

Declarative Representations.

(1) Semantic Net
(2) Conceptual dependency
(3) Frames.
(4) Scripts.

(1) Semantic Net. It was originally developed to represent the meanings of English words. In semantic net the informations are represented as set of nodes, connected to each other, by a set of labelled arcs which represent the relationship among nodes.

Example



AI Languages : Lisp : List Processing.

Handles list.

Prolog : Programming Logic

Predicate logic resolution.

(iv) Semantic Net can be easily represented in Lisp:

Atom	Properties.
Chair	((is a Furniture))
Mychair	((is a chair) (coloured tan) (Covering of leather) (owner me))

(v) Using Predicate —

Two Place Predicate:

Is_a(Chair, Furniture).

Man(Marcus) is rewritten as Is_a(Marcus, Man).

Conceptual Dependency (C.D)

It is a theory for how to represent the meaning of natural language sentence in way that —

(i) Facilitate drawing influence from the sentences.

(ii) Independent of the language in which the sentences were originally stated.

The C.D. representation is built out of conceptual primitives that can be combined to form the necessary words in any particular language.

It also provides specific structures and set of primitives, out of which representation of particular pieces of information can be constructed.

Primitives.

- ATRANS → Transfer of an abstract relationship. (Eg: give).
- PTRANS → Transfer of physical location of an object (Eg: go).
- Propel → Application of physical force on an object (Eg: push).
- Move → Movement of body parts.
- Grasp → Grasping an object by an actor (Eg: Throw)
- INGEST → Taking an object by an animal (Eg: Eat).
- Expel → Expulsion of something from the body of an animal into the world. (Eg. cry).

Example: I gave the man a book.

O → object case relation

→ Indicates the direction of dependency

↔ Two way link b/w actor & action

P Past tense

R Recipent case relation.

Four Primitive Conceptual Categories [Second Set of Building Blocks]

ACT's Action

PP's Object (Picture Producer)

AA's Modifiers of Action (Action Aides)

PA's Modifier of PP's.

There are set of rules —

Example: John ↔ PTRANS

John Ran.

- PP ↔ ACT.

Scripts

It provides a structure that describes a stereotypical sequence of events in a context.

A script consists of a set of slots and associated with each slot, some information about the kind of values, along with a default value

Important Components of a Script

- Entry Condition: The condition that must be satisfied before the event in the script, can occur.
- Result: The condition that will become true after events that describe the script, have occured.
- Props: Slots that represent the objects that are involved in the Scripts.
- Roles: Slots the represent the people who are involved in the scripts described in the script.

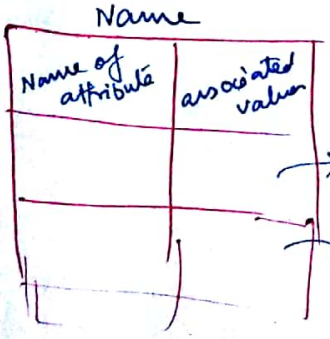
- Track : Specific variation on a more general pattern that is represented by the particular scripts
- Scene : Actual sequence of events that occur

28/11 Example of Scripts

Script: Restaurant Track: Coffee Shop Props: Table, Chairs, Menu Roles: S – Customer, W – Waiter, C – Cook, M – Cashier, O – Owner	Scene 1: Entering S. PTRANS S. ATTEND
	Scene 2: Ordering
Entry-Condition: S. Hungry S. Has_Money	Scene 3: Eating
Result: S. has_less_money S. is_not_hungry O. has_more_money S. is_satisfied (optim)	Scene 4: Exiting

Frames

Set of attributes (slots) and associated values.
Used to describe class and instance of the class (objects).



→ can be default value if nothing present

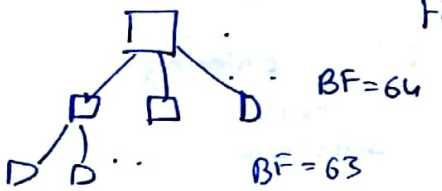
→ attached procedure will be invoked
[can be inherited to other attributes if needed]

Constraint Satisfaction Problem

Problem solving on satisfying the constraints present in a specific problem.

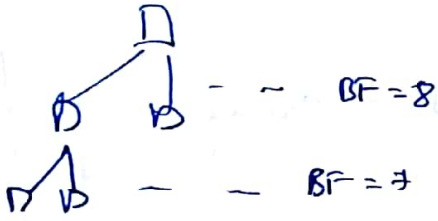
Example : Satisfying constraints : No shared rows, columns, diagonals for N-Queens problem.

For N Queens problem —



D D . . . D → BF=57 leaf for 8 Queen.

If row constraint applied,



Now if all the constraints are applied, the BF greatly reduces.