EX.NO : 8

UNIFICATION AND RESOLUTION

AIM:

To execute programs based on Unification and Resolution.Deduction in prolog is based on the Unification and Instantiation. Matching terms are unified and variables get instantiated.

Example 1: In the below prolog program , unification and instantiation take place after querying.

Facts :

likes(john, jane).

likes(jane, john).

Query :

?- likes(john, X).

Answer : X = jane.

Here upon asking the query first prolog start to search matching terms in predicate with two arguments and it can match likes(john, ...) i.e.Unification. Then it looks for the value of X asked in query and it returns answer X = jane i.e.Instantiation - X is instantiated to jane.

Example 2 : At the prolog query prompt, when you write below query,

?- owns(X, car(bmw)) = owns(Y, car(C)).

You will get Answer : X = Y, C = bmw.

Here owns(X, car(bmw)) and owns(Y, car(C)) unifies – because

(i) predicate names are same on both side

(ii) number of arguments for that predicate, i.e. 2, are equal both side.

(iii) 2nd argument with predicate inside the brackets are same both side and even in that predicate again number of arguments are same. So, here terms unify in which X=Y. So, Y is substituted with X -- i.e. written as {X | Y} and C is instantiated to bmw, -- written as {bmw | C} and this is called Unification with Instantiation.

But when you write ?- owns(X, car(bmw)) = likes(Y, car(C)). then prolog will return False, since it can not match the ;owns; and ;likes; predicates.

Resolution is one kind of proof technique that works this way –

(i) select two clauses that contain conflicting terms

(ii) combine those two clauses and

(iii) cancel out the conflicting terms.

For example we have following statements,

(1) If it is a pleasant day you will do strawberry picking
(2) If you are doing strawberry picking you are happy.

Above statements can be written in propositional logic like this -

(1) strawberry_picking ← pleasant
(2) happy ← strawberry_picking

And again these statements can be written in CNF like this -

(1) (strawberry_picking ∨~pleasant) ∧
(2) (happy ∨~strawberry_picking)

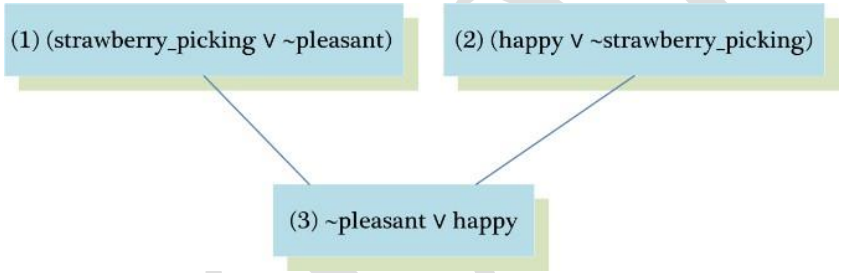
By resolving these two clauses and cancelling out the conflicting terms ;strawberry_picking; and ;~strawberry_picking;, we can have one new clause,

(3) ~pleasant ∨ happy

How ? See the figure on right.

When we write above new clause in infer or implies form, we have ;pleasant → happy; or ;happy ← pleasant;

i.e. If it is a pleasant day you are happy.



But sometimes from the collection of the statements we have, we want to know the answer of this question - "Is it possible to prove some other statements from what we actually know?" In order to prove this we need to make some inferences and those other statements can be shown true using Refutation proof method i.e. proof by contradiction using Resolution. So for the asked goal we will negate the goal and will add it to the given statements to prove the contradiction.

Let;s see an example to understand how Resolution and Refutation work. In below example, Part(I) represents the English meanings for the clauses, Part(II) represents the propositional logic statements for given english sentences, Part(III) represents the Conjunctive Normal Form (CNF) of Part(II) and Part(IV) shows some other statements we want to prove using Refutation proof method.

Part(I) : English Sentences

(1) If it is sunny and warm day you will enjoy.
(2) If it is warm and pleasant day you will do strawberry picking

(3) If it is raining then no strawberry picking.
(4) If it is raining you will get wet.
(5) It is warm day
(6) It is raining
(7) It is sunny

Part(II) : Propositional Statements

(1) enjoy ← sunny ∧ warm
(2) strawberry_picking ← warm ∧ pleasant
(3) ~strawberry_picking ← raining
(4) wet ← raining
(5) warm
(6) raining
(7) sunny

Part(III) : CNF of Part(II)

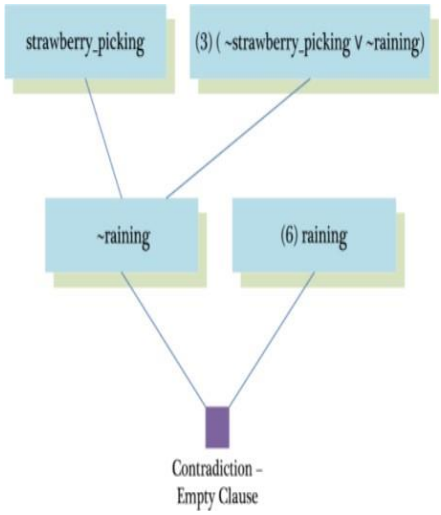
(1) (enjoy ∨~sunny∨~warm) ∧
(2) (strawberry_picking ∨~warm∨~pleasant) ∧
(3) (~strawberry_picking ∨~raining) ∧
(4) (wet ∨~raining) ∧
(5) (warm) ∧
(6) (raining) ∧
(7) (sunny)

Part(IV) : Other statements we want to prove by Refutation

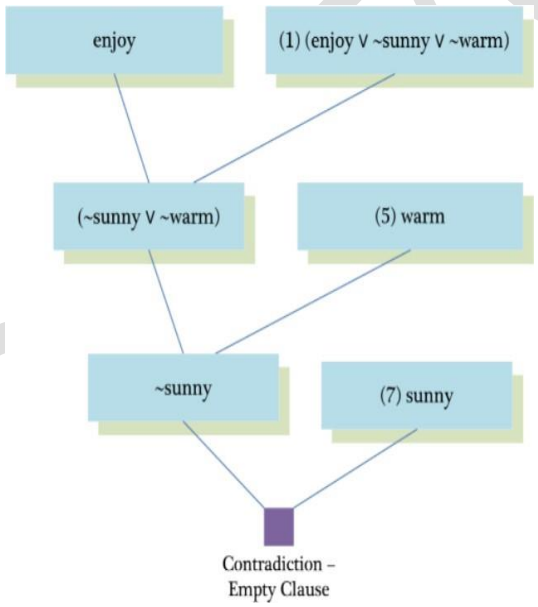
(Goal 1) You are not doing strawberry picking.
(Goal 2) You will enjoy.
(Goal 3) Try it yourself : You will get wet.

Goal 1 : You are not doing strawberry picking.

Prove : ~strawberry_picking
Assume : strawberry_picking (negate the goal and add it to given clauses).



Goal 2 : You will enjoy.
Prove : enjoy
Assume : ~enjoy (negate the goal and add it to given clauses)



SOURCE CODE:

```
enjoy:-sunny,warm.
strawberrry_picking:-warm,plesant.
notstrawberry_picking:-raining.
wet:-raining.
warm.
raining.
sunny.
```

OUTPUT:

```
?- notstrawberry_picking.
true.

?- enjoy.
true.

?- wet.
true.
```


RESULT:

Thus the python code is implemented successfully and the output is verified.