

UNIT-IV: Declarative Programming

Paradigm: Logic Programming



Faculty In-charge

Mrinmoyee Mukherjee

Assistant Professor (IT Dept.)

email: mrinmoyeemukherjee@sfit.ac.in

Mob: 9324378409

Academic Year: 2023-24



OUTLINE OF UNIT-4

Sub-Unit

Contents

- 4.0 Logic programming with PROLOG
 - 4.1 Resolution and Unification
 - 4.2 Lists, Arithmetic execution order
 - 4.3 Imperative control flow
 - 4.4 Database manipulation
 - 4.5 PROLOG facilities and deficiencies
-



4.0: Logic Programming with PROLOG



INTRODUCTION

- Distinctive style of Programming
- Collection of logical propositions and questions. There are only two components to any program: **facts and rules**.
 - The program is a knowledge base of facts and a series of rules to be applied to knowledge base
 - Programs are based on the techniques developed by logicians to form valid conclusions from available evidence (knowledge base)
- Interaction with the program: Posing queries to an inference engine (also called a query interpreter)
- Most widely used is Prolog. The name stands for Programming in Logic



Introduction to Logic Programming: Prolog

- Prolog is based on Horn Clauses.
- Horn Clauses are subset of Predicate Logic
- Predicate Logic is a syntax for reading and writing logical ideas
- Predicate Logic defines how reasoning gets done in logic terms
- To transform English statement to Predicate Logic, we need to remove unnecessary terms. What remains is relationship and entities or arguments
- English Statement: Elephant is bigger than Horse
 - Here the relation is bigger, and entities involve are Elephant and Horse
 - The relation is also called as Functor
- Prolog has three different types of clauses
 - Facts, Rules and Questions (or Queries)

Relation

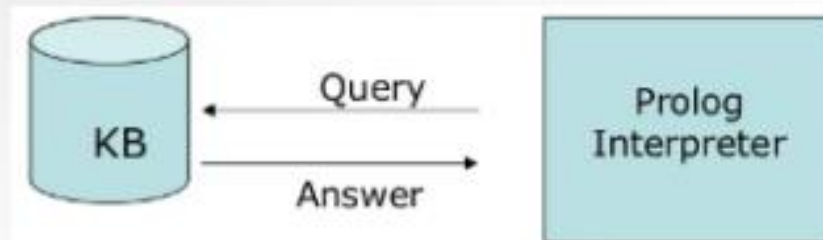
bigger(elephant, horse).

Arguments

Fact, There is full stop at the end
The entities are given in small case letters

Introduction to Prolog

- Every clause has full stop at the end.
- Clauses that are always True, are called as **Facts**
 - Ex. father(ali, haya).
- Some clauses are dependent on other clauses for being True, these are called as **Rules**
 - Ex. parent(person1, person2) :- father(person1, person2).
- To run a program in Prolog, we ask **Questions** about database



- Key Features of Prolog :
 - **Unification** : The basic idea is, can the given terms be made to represent the same structure.
 - **Backtracking** : When a task fails, prolog traces backwards and tries to satisfy previous task.
 - **Recursion** : Recursion is the basis for any search in program.

Introduction to Prolog

- **Advantages :**

1. Easy to build database. Doesn't need a lot of programming effort.
2. Pattern matching is easy. Search is recursion based.
3. It has built in list handling. Makes it easier to play with any algorithm involving lists.

- **Disadvantages :**

1. LISP (another logic programming language) dominates over prolog with respect to I/O features.
2. Sometimes input and output is not easy.

- **Applications :**

1. Prolog is highly used in artificial intelligence(AI).
2. Prolog is also used for pattern matching over natural language parse trees.

Introduction-PROLOG

FACTS

OBJECTS

RELATIONSHIP

- Olivia likes Pizza.

likes(olivia, pizza).

FACTS

RELATIONSHIP

OBJECTS



Introduction-PROLOG

FACTS

OBJECTS

RELATIONSHIP

- Olivia likes Pizza.

likes(olivia, pizza).

FACTS

RELATIONSHIP

OBJECTS

- Beginning with lower case letters.
- End the fact with period (full stop).



Introduction-PROLOG



Clauses



Knowledge Base



Prolog programming is all about writing knowledge base. We fire the query on the knowledge base



Introduction-PROLOG -FACTS

- Facts are statements about what is true about a problem, instead of instructions how to accomplish the solution.
- The Prolog system uses the facts to work out how to accomplish the solution by searching through the space of possible solutions.
- It is defined by an identifier followed by an n-tuple of constants.
- A relation identifier is referred to as a predicate
- When a tuple of values is in a relation we say the tuple satisfies the predicate.

SYNTAX

- Names of relationship and objects must begin with a lower- case letter.
- Relationship is written *first* (typically the *predicate* of the sentence).
- *Objects* are written separated by commas and are enclosed by a pair of round brackets.
- The full stop character ‘.’ must come at the end of a fact.



EXAMPLES

valuable(gold)

Gold is Valuable

owns(john, gold)

John owns gold

father(john, mary)

John is the father of Mary

gives (john, book, mary)

John gives the book to Mary



Introduction-PROLOG -RULES

- Specifies under what conditions a tuple of values satisfies a predicate.
- The basic building block of a rule is called an *atom*
- $\text{Atom} :- \text{Atom1}, \dots, \text{Atomn}$
- *If each of Atom1,...,Atomn is true, then Atom is also true.*



Introduction-PROLOG -KNOWLEDGE BASE(CLAUSES)

- reads_a_book(saeed).
 - happy(khalid).
- HEAD OF CLAUSE NECK OF CLAUSE BODY OF CLAUSE
- FACTS
- author(saeed):- reads_a_book(saeed).
 - reads_a_book(khalid):- happy(khalid).
 - author(khalid):- reads_a_book(khalid).
- RULES

part on the left hand of the “ :- ” is called the head
part on the right hand of the “ :- ” is called the body;

rule->

head :- body

General Rule: If the body of the rule is true then head implies true



BASIC EXAMPLE

```
dog(fido) .  
dog(rover) .  
dog(henry) .  
cat(felix) .  
cat(michael) .  
cat(jane) .  
animal(X) :- dog(X) .
```

FACT: Interpretation that fido, rover and henry are all dogs.

FACT: Interpretation that felix, michael and jane are all cats

RULE: Saying that anything (let us call it X) is an animal if it is a dog.

- Cat lovers may feel that cats can also claim to be called animals, but the program is silent about this.
- Load the program
- You will come to system prompt, symbol ?-
- To check whether fido is a dog all that is necessary is to type the query `dog(fido)` followed by a full stop and press the 'return' key

?- dog(fido).



BASIC EXAMPLE continued.....

?-dog(jane).

no

[Is jane a dog? No - a cat]

?- animal(fido).

yes

[Is fido an animal?]

[yes - because it is a dog and any dog is an animal]

?- dog(X).

X = fido ;

X = rover ;

X = henry

[Is it possible to find anything, let us call it X, that is a dog?]

[All 3 possible answers are provided]

?-animal(felix).

no

[felix is a cat and so does not qualify as an animal, as far as the program is concerned]

- Example shows the two components of any Prolog program, rules and facts
- The use of queries that make Prolog search through its facts and rules to work out the answer.



BASIC EXAMPLE continued.....

?-dog(jane).
no

[Is jane a dog? No - a cat]

?- animal(fido).
yes

[Is fido an animal?]
[yes - because it is a dog and any dog is an animal]

?- dog(X).
X = fido ;
X = rover ;
X = henry

[Is it possible to find anything, let us call it X, that is a dog?]

[All 3 possible answers are provided]

?-animal(felix).
no

[felix is a cat and so does not qualify as an animal, as far as the program is concerned]

GIVEN THAT
any X is an animal if it is a dog

AND
fido is a dog

DEDUCE
fido must be an animal



Introduction-PROLOG -EXAMPLE

FACTS	ENGLISH MEANING
food(burger).	// burger is a food
food(sandwich).	// sandwich is a food
food(pizza).	// pizza is a food
lunch(sandwich).	// sandwich is a lunch
dinner(pizza).	// pizza is a dinner
RULES	
meal(X) :- food(X).	// Every food is a meal OR Anything is a meal if it is a food
QUERIES/GOALS	
?- food(pizza).	// Is pizza a food?
?- meal(X), lunch(X).	//Which food is meal and lunch?
?- dinner(sandwich).	//Is sandwich a dinner



DATA OBJECTS IN PROLOG

NUMBERS

- All versions of Prolog allow the use of integers (whole numbers).
- They are written as any sequence of numerals from 0 to 9
- Optionally preceded by a + or – sign

Ex: 623, -47, +5, 025

- Most versions of Prolog also allow the use of numbers with decimal points.
- They are written in the same way as integers, but contain a single decimal point, anywhere except before an optional + or – sign

Ex: 6.43, -.245, +256.



DATA OBJECTS IN PROLOG continued...

ATOMS

- Atoms are constants that do not have numerical values. There are three ways in which atoms can be written

john
today_is_Tuesday
fred_jones
a32_BCD

but not

Today
today-is-Tuesday
32abc

'Today is Tuesday'
'today-is-Tuesday'
'32abc'

Any sequence of characters enclosed in single quotes, including spaces and upper case letters, e.g.

Examples

+++
>=
>
+--

Any sequence of one or more special characters from a list that includes the following + - * / > < = & # @ :

Any sequence of one or more letters (upper or lower case), numerals and underscores, beginning with a lower case letter



DATA OBJECTS IN PROLOG continued...

VARIABLES

- In a query a variable is a name used to stand for a term that is to be determined
- The name of a variable is denoted by any sequence of one or more letters (upper case), numerals and underscores, beginning with an upper case letter or underscore, e.g.

X
Author
Person_A
_123A

but not

45_ABC
Person-A
author



4.1: Resolution and Unification

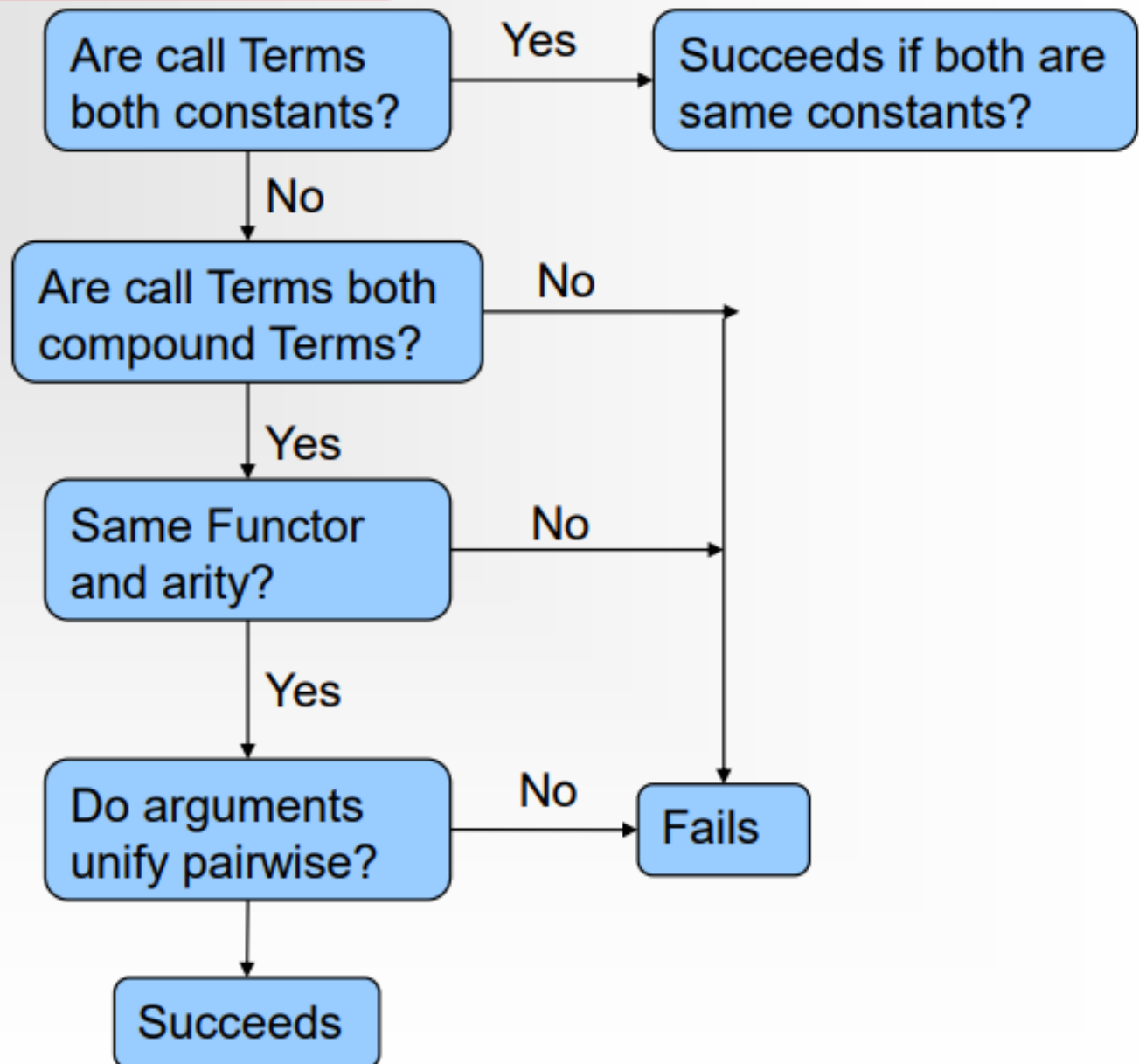
Unification

- When a goal is given to evaluate, Prolog will work through the clauses in the database.
- It attempts to match the goal with each clause.
- The matching process works from left to right.
- The goal will fail if no match is found. If a match is found, the action will be taken.
- Prolog uses a matching technique called unification technique
 - Here one or more variables being given value to make the two call terms identical.
 - This process is called binding the variables to values.
 - For example, Prolog can unify the terms `cat(A)`, and `cat(mary)` by **binding variable A to atom mary** that means we are giving the value mary to variable A

UNIFICATION

- Definition: Unification is the process in which one or more variables being given values in order to make two call terms identical. This is known as binding the variables to values.
- `dog(X)` and `dog(fido)`
Can be unified by binding variable `X` to atom `fido`, giving `X` the value `fido`
- `owns(john,fido)` and `owns(P,Q)`
Can be unified by binding variables `P` and `Q` to atoms `john` and `fido`, respectively.
- `parent(albert, adward)` and `parent(victoria, Y)`
Cannot be unified as `albert` and `victoria` do not unify

Unification Algorithm



Unification

- **Unification** is like assignment
- It is kind of like equality
- It is also like pattern matching
- The explicit unification operation is =
- Any **value** can be unified with **itself** E.g. Mary=Mary
- Any **variable** can be unified with other **variable** E.g. X=Y
- A **variable** can be unified with any **other Prolog value**

E.g. **Season** = weather(sunny).

Here variable **Season** is instantiated to value weather(sunny)

- Thus, **two terms unify** if they are **same** term **or** if they contain **variable** that can be uniformly **instantiated with** the **term** in such a way that the resulting terms are equal

Unification

- Two different structures can be unified if their constituents can be unified

`female(X) = female(jane).`

Here X is unified with jane

- A variable can be unified with a structure containing that same variable
- Look at the following example
 - `mother(mary, X) = (Y, father(Z)).`
 - means `X = father(Z)`

- Following examples of no unification

`vincent` and `mia` do not unify

`woman(mia).` and `woman(jody).` will not unify

Unification

- **Unification** is **symmetric** means

Steve = father(isaac) **is same as** father(isaac) = Steve

- In well written code most **unification happens implicitly** as a result of **parameter matching**.
- Suppose in database there is a fact..
teaches(**john**, nora).
- If we query the database as..
?-teaches(**X**, nora).
- X and john are unified
- When Prolog unifies two terms, it performs all the necessary instantiations, so that the terms are equal afterwards

Unification Examples

?-mia = mia

yes

?-43 = 43

yes

?-mia=vincent

no

?-mia=X

X=mia

?-X=mia,

X=Vincent

no

?- k(s(g), Y) = k(X, t(k)).

X = s(g)

Y = t(k)

Yes

?- k(s(g), t(k)) = k(X, t(Y)).

X = s(g)

Y = k

Yes

?- vertical(line(point(X,Y), point(X,Z))).

?- horizontal(line(point(X,Y), point(Z,Y))).

?-vertical(line(point(1,1), point(1,3))).

yes

?-vertical(line(point(1,1), point(3,2))).

no

?- horizontal(line(point(1,1), point(1,Y))).

Unification Examples

Ex1

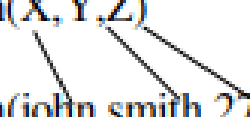
- Facts :
likes(john, jane).
likes(jane, john).
- Query :
?- likes(john, X).
Answer : $X = \text{jane}$.
- Here upon asking the query first prolog start to search matching terms in 'Facts' in **top-down manner** for 'likes' predicate with two arguments and it can match likes(john, ...) i.e. **Unification**.
- Then it looks for the **value of X** asked in query and it returns answer $X = \text{jane}$ i.e. **Instantiation** - X is instantiated to 'jane'.

Ex2:

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

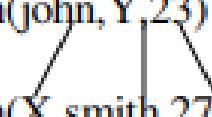
You will get Answer : $X = Y, C = \text{bmw}$.

person(X,Y,Z)
person(john,smith,27)



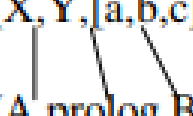
Succeeds with variables X , Y and Z bound to *john*, *smith* and 27, respectively.

person(john,Y,23)
person(X,smith,27)



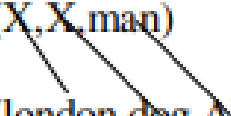
Fails because 23 cannot be unified with 27

pred1(X,Y,[a,b,c])
pred1(A,prolog,B)



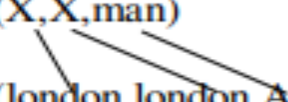
Succeeds with variables X and A bound to each other, Y bound to atom *prolog* and B bound to list $[a,b,c]$.

pred2(X,X,man)
pred2(london,dog,A)



Fails because X cannot unify with both the atoms *london* and *dog*.

pred3(X,X,man)
pred3(london,london,A)



Succeeds with variables X and A bound to atoms *london* and *man*, respectively.

RESOLUTION

- Resolution in Prolog is basically the inference mechanism
- Means, when you resolve two clauses you get one new clause.
- (1) *All women like shopping.* (2) *Olivia is a woman.* Now we ask query 'Who likes shopping'. So, by resolving above sentences we can have one new sentence *Olivia likes shopping.*
- Formally a clause is defined as an expression of the form
- $(A_1 A_2 \dots A_n) \mid (B_1 B_2 \dots B_n)$
- Where each of the As and Bs stand for proposition
-
- A clause is used to express a logical implication of the following sort-
- -If all of Bs are true then one of the As must be true as well
- Resolution in Prolog is basically the inference mechanism

Resolution

- **Resolution** is inference mechanism.
- Derivation of new statement is called **Resolution**
- Consider two clauses

$m :- b.$

$t :- p, m, z.$

- So from that we can infer

$t :- p, b, z.$

- that is called resolution.
- Means, when you resolve two clauses you get one new clause.
- Another example, we have two sentences

All women like shopping.

Olivia is a woman.

- Now we ask query 'Who likes shopping'.
- So, by resolving above sentences we can have one new sentence

Olivia likes shopping.

$C :- A, B$

A and B imply C

$D :- C$

$D :- A, B$

If we know that A and B imply C,
and that C implies D,
then we can deduce that A and B imply D

Resolution Examples

flowery(X) :- rainy(X).

Predicate applied to a variable

rainy(Mumbai).

Predicate applied to atom

- From above two clauses we can infer third clause

flowery(Mumbai).

```
?- pwd.
```

```
% d:/documents/prolog/
```

```
true.
```

To check current path

```
?- cd('D:/PCPF/AY-2023-24/Course Lab/Prolog').
```

```
true.
```

To change current path

```
?- pwd.
```

```
% d:/pcpf/ay-2023-24/course lab/prolog/
```

```
true.
```

```
?- |
```



Type here to search



prolog1.pl - Notepad
File Edit Format View Help

```
woman(mia).  
woman(jody).  
woman(yolanda).  
playsairguitar(jody).  
men(bheem).  
men(chotu).  
playscricket(chintu).  
  
% d:/pcpf/ay-2023-24/course lab/prolog/  
true.  
  
?- [prolog1].  
true.  
  
?- woman(mia).  
true.  
  
?- woman(jody).  
true.
```

?- woman(pia).

false.

?- women(mia).

Correct to: "woman(mia)"? yes

true.

?- playsairguitar(jody).

true.

?- men(chotu).

true.

?- men(chotu1).

false.

?-

```
happy(yolanda).  
happy(jerry).  
happy(tom).  
sad(bheem).  
sad(chutki).  
listens2music(mia).  
listens2music(yolanda):-happy(yolanda).  
playsairguitar(mia):-listens2music(mia).  
playsairguitar(yolanda):-listens2music(yolanda).
```

true.

?- happy(tom).

true.

?- sad(bheem).

true.

?- sad(team).

false.

?- happy(X).

X = yolanda .

?- happy(X).

X = yolanda ;

X = jerry ;

X = tom.

?- |

X is 3 .

?- X is 3.

X = 3.

?- X is 17.

X = 17.

?- X is sqrt(26).

X = 5.0990195135927845.

?- X is sqrt(36).

X = 6.0.

?- X is sqrt(36)+2.

X = 8.0.

?- sqrt(36)+4=:5*11-45.

true.

?- |

END