Artificial Intelligence Programming in Prolog
Lecture-1
Introduction to PROLOG

Dr. Nasima Begum
Asst. Prof., Dept. of CSE, UAP

References

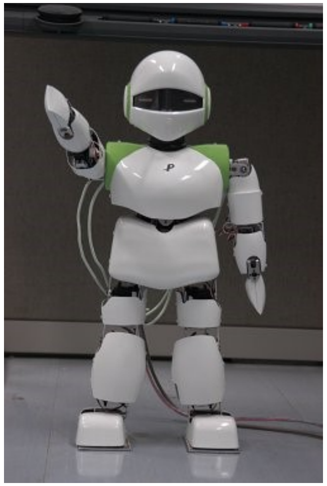
- Useful references:
  - Clocksin, W.F. and Mellish, C.S., Programming in Prolog: Using the ISO Standard (5th edition), 2003.
  - Bratko, I., Prolog Programming for Artificial Intelligence (3rd edition), 2001.
  - Sterling, L. and Shapiro, E., The Art of Prolog (Second edition), 1994.

PROLOG

- Prolog is a logic programming language associated with artificial intelligence and computational linguistics.
- **First appeared:** 1972
- **Designed by:** Alain Colmerauer, Robert Kowalski

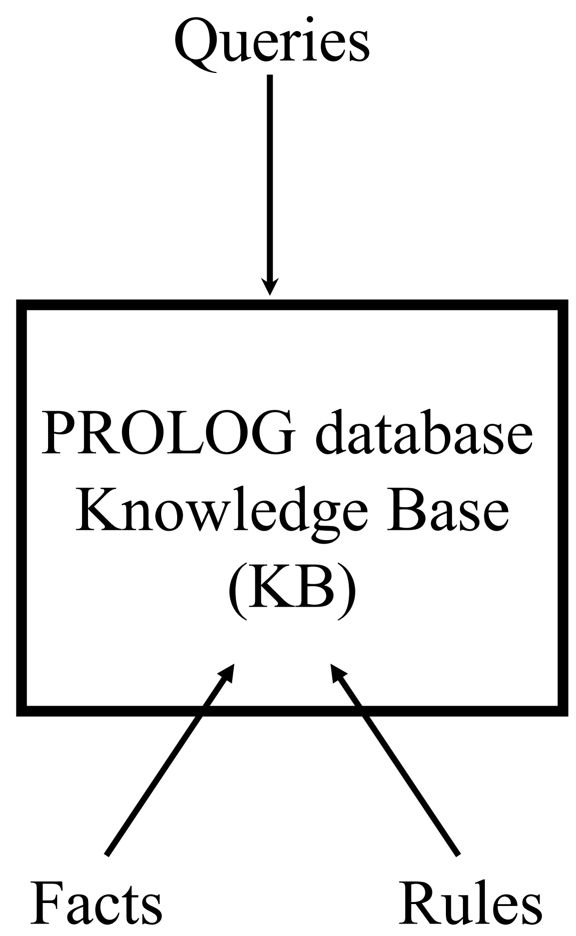
PROLOG

- PROLOG means PROgramming in LOGic
  - The programmer uses the system to draw inferences from facts and rules
- PROLOG is
  - declarative - specify facts and logical relationships
  - symbolic - symbols are used to represent objects
  - high level - contains a built-in problem solving mechanism
- PROLOG Programs
  - solve problems by declaring objects and their relationships



PROLOG Paradigm

- The PROLOG Programmer
  - Loads facts and rules into the database.
  - Makes queries to the database to see if a fact is:
    - in the database or
    - can be implied from the facts and rules therein



Facts:

isa(dog, mammal).

isa = **predicate**

(dog, mammal) = **argument/atom**

2 = **binary**

PROLOG Paradigm … …

- Predicates (clauses)

  In this example, both "isa" and "animal" are examples of predicates.

- Predicates are used to indicate relations between objects and hence can represent FACTS and RULES.

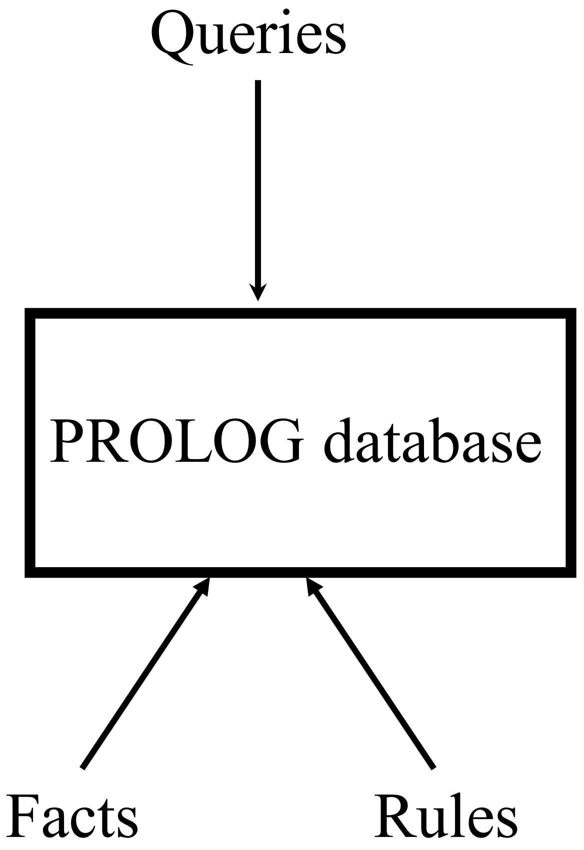
- Two ways to use predicates in a query:
  - 1. As a TRUE/FALSE test:

    ?- isa(dog, mammal).

    yes
  - 2. For DATA RETRIEVAL

    ?- isa(dog, X).

    X = mammal



PROLOG syntax

- Constants
  - **Atoms**
    - Alphanumeric atoms - alphabetic character sequence starting with a lower case letter

    Examples: apple a1 apple_cart

    - Quoted atoms - sequence of characters surrounded by single quotes

    Examples: 'Apple' 'hello world'

    - Symbolic atoms - sequence of symbolic characters

    Examples: & < > * - + >>

    - Special atoms

    Examples: ! ; [ ] {}
  - **Numbers**
    - Integers and Floating Point numbers

    Examples: 0 1 9821 -10 1.3 -1.3E102

PROLOG syntax … …

- Variable Names

  A sequence of alphanumeric characters beginning with an upper case letter or an underscore

  Examples: Anything _var X

- Compound Terms (structures)
  - an atom followed by an argument list containing terms. The arguments are enclosed within brackets and separated by commas

  Example: isa(dog, mammal)

System Interaction

Reminder:

Problem solving in PROLOG

- 1. insert facts and rules into the knowledge-base (KB)
- 2. ask questions (queries) based on the contents of the knowledge-base (KB)

- Facts
  - Used to represent unchanging information about objects and their relationships.
  - Only facts in the PROLOG database can be used for problem solving.
  - Insert facts into the knowledge-base by,
    - typing the facts into a file, save the file with .pl extension and loading (**consulting**) the file into a running PROLOG system

System Interaction … …

- Queries
  - Retrieve information from the knowledge-base by entering QUERIES
  - A query,
    - is a pattern that PROLOG is asked to *match* against the database
    - has the syntax of a compound query
    - may contain variables
  - A query will cause PROLOG to
    - look at the database
    - try to find a match for the query pattern
    - execute the **body** of the matching **head**
    - return an answer

System Interaction … …

Example:

Assume that the knowledge-base contains:

- `likes(hasan, rita).`
- `likes(belal, rita).`
- `likes(mohsin, beli).`

The actual system interaction looks like

```
?- likes(hasan, rita).
yes
?- likes(mohsin, rita).
no
?- likes(mohsin, X).
X = beli;
```

Variables in Prolog

Reminder: a variable starts with a capital letter or underscore.

When a variable is used, PROLOG tries to find a match (*instantiation*) for it.

```
?-likes(Who, rita).
Who = hasan;
Who = belal;
no
```

Simple Backtracking

- PROLOG searches its database attempting to satisfy a query (goal), stopping at the first success or returning no for failure.
- Often, there will be more than one successful match and the programmer would like to tell the PROLOG system to try again and search for other successful matches.
- When PROLOG **retries a goal**, it is called **backtracking**.
- Backtracking may be forced by typing a **semicolon (;)**

Examples: Backtracking

```
?- likes(Who, rita).
Who = hasan ;
Who = belal ;
no
?- likes(Who, Whoelse).
Who = hasan
Whoelse = rita ;
Who = belal
Whoelse = rita ;
Who = mohsin
Whoelse = beli ;
no
```

```
?- likes(Who, beli).
Who = mohsin
?- likes(hasan, Whom).
Whom = rita ;
no
?- likes(belal, Whom).
Whom = rita ;
no
```

Backtracking with Anonymous Variable

Suppose that you want to know if Hasan likes anyone and do not care who in particular.

Then use the anonymous variable which is the character "_"

It represents a different variable each time it occurs

```
?- likes(hasan, _).
yes
?- likes(_, _).
yes
```

Suppresses output of variable binding

Rules

- A PROLOG rule consists of **one conclusion** followed by **one or more conditions**
  - a fact is a rule with no conditions
  - **conclusion :-**

    condition1,

    .....

    condition N.
  - read as:
    - The conclusion is true if condition1 and condition2... and condition N are true.
    - Conditions are separated by commas
    - Rules, facts and queries are all examples of Horn clauses

Rules …

- A PROLOG rule consists of a head and a body
  - head :-

    body.
  - read as:
    - If a match is made on the head, then carry out the body.

Arithmetic Functions/ Predicates

- Operators for the basic arithmetic operations

  +, -, *, /, **(power), mod, // (integer division).

  Examples:

  ?- 7+3=10.

  No

  ?- 10 is 7+3.

  Yes

  ? - X is 5/2. (=2.5)

  ? - Y is 17//3. (=5)

  ?- Z is 23 mod 5. (=3)

- Predicates for the basic arithmetic operations

  <, =<, >, >=, =:= (equal), =\= (not equal), ==(identical), \==(not identical).

  Examples:

  ?- 14-9=:=3+2.

  Yes

  ? - 10 =\=7+5.

  Yes

Arithmetic Functions/ Predicates

- Arithmetic Expression Equality E1=:=E2 succeeds if the arithmetic expressions E1 and E2 evaluate to the same value.
- Example:

?- 6+4=:=6*3-8.

yes

?- sqrt(36)+4=:=5*11-45. (=10)

yes

- Arithmetic Expression Inequality =\= E1=\=E2 succeeds if the arithmetic expressions E1 and E2 do not evaluate to the same value.

?- 10=\=8+3.

Yes

Arithmetic Functions/ Predicates

- TERM IDENTICAL == Both arguments of the infix operator == must be terms. The goal Term1==Term2 succeeds if and only if Term1 is identical to Term2. Any variables used in the terms may or may not already be bound, but no variables are bound as a result of evaluating the goal.

```
?- likes(X,prolog)==likes(X,prolog).
true.
?- likes(X,prolog)==likes(Y,prolog).
false.
?- 6+4==3+7.
false.
?- 6+4==6+4.
true.
```

```
?- a+b==b+a.
false.

?- a+b==a+b.
true
```

Arithmetic Functions/ Predicates

- Terms Not Identical \== Term1\==Term2 tests whether Term1 is not identical to Term2. The goal succeeds if Term1==Term2 fails. Otherwise it fails.

```
?- pred1(X)\==pred1(Y).
   true.
?- 5+3\==8.
true.
```

- (The output signifies that both X and Y are unbound and are different variables.)

PROLOG Syntax … …

- Lists

A sequence of terms of the form

$[t_1, t_2, t_3, t_4, ..., t_n]$

where term t_i is the i^{th} element of the list

Examples:

- [a,b,c] is a list of three elements a, b and c.

- [[a,list,of,lists], and, numbers,[1,2,3]]

is a four element list.

- **[ ] is the 'empty list'. It is an atom not a list data type**