Wan Hussain Wan Ishak
School of Computing
UUM College of Arts and Sciences
Universiti Utara Malaysia

(P) 04-9285150
(E) hussain@uum.edu.my
(U) http://wanhussain.com
Lecture notes

Overview of Prolog

- Main components
- Syntax
- Terms
- Pattern matching, unification and instantiation
- Backtracking
- Arithmetic Operator
- Linguistic Operator
Introduction to logic...

- A form of knowledge representation.

- Study on reasoning process and production system (also rule) that support the reasoning process.

Premise
- Fact
- Sentence
- Observation
- Information

Logical Process

Inference
- New fact
- Action
- Conclusion
The idea of Prolog

- A programming language.

“PROgramming in LOGic” $\Rightarrow$ PROLOG

- Very versatile language – can do or implement all kinds of algorithms.
The idea of Prolog

Prolog program consist of

1) a set of facts
2) a set of conditions

The computer can figure out for itself how to deduce the solution from the facts given.
The idea of Prolog

- Invented by Alain Colmerauer and his colleagues at the University of Aix-Marseille, France in 1972.

- Powerful language for AI and non-numerical programming in general.

- Commercially used in expert systems, intelligent databases, and natural language processing programs.
Varieties of Prolog

- Many Prolog versions are available.

- Standard applied are the same – different in syntax, built-in function, and operating system compatibility.

Example:

- Arity Prolog
- Quintus Prolog
- Amzi Prolog
- ALS Prolog
- SWI Prolog
- LPA Prolog
Prolog vs Lisp

- Both easy to perform complex computations on complex data.
- Both allocate memory dynamically – programmer does not have to declare the size of data structures before creating them.
- Both can examine and modify itself.
Prolog vs Lisp

What difference?

Prolog has an automated reasoning procedure – an INFERENCE ENGINE.

Programs that perform logical reasoning are much easier to write in Prolog.
How it works?

Process - Procedural Interpretation of Logic.

- Knowledge is represented in terms of procedure definitions – clauses.
- Reasoning – a process of calling the right procedures.
How it works?

Example:

[1] For any X, if X is in Kedah, then X is in the Malaysia
[2] Alor Star is in Kedah

Note:

[1] is a RULE – enables us to infer other info.
[2] is a fact – does not depend on other
How it works?

To know whether Alor Star is in Malaysia – [1] and [2] can be chained together.

In Prolog:

```prolog
in_Malaysia(X):- in_Kedah(X).

in_Kedah(alor_star).
```

Note:
in_Malaysia and in_Kedah are PREDICATES
Declarative meaning

- concerned only with the **relations** defined in the program.
- determines **what** will be the output of the program.

Procedural meaning

- How the output is obtained – how the relations are actually evaluated by Prolog.
Terminology

- Fact
- Rule
- Predicate
- Argument
- Arity

Consists of a predicate with or without an argument.

Example:

state(kedah).
wan_hussain.
Terminology

- Fact
- Rule
- Predicate
- Argument
- Arity

Predicate that depend on other predicates/facts or information.

Example:

\[
\text{Is\_in}(X,Y):-
\text{state}(X),
\text{country}(Y,X).
\]
Terminology

- Fact
- Rule
- Predicate
- Argument

Consists of name, bracket, and arguments

Example:

country(malaysia).
Terminology

- Fact
- Rule
- Predicate
- Argument
- Arity

Element in predicate (written in bracket)

Example:
For country(malaysia) the argument is ‘malaysia’.
Terminology

- Fact
- Rule
- Predicate
- Argument

Arity

Referring to the number of arguments in predicate.

Example:

state(malaysia).
   → state/1

father(ahmad, karim).
   → father/2

foo.
   → foo/0
Arity

- Two distinct predicates can have the same name if they have different arities.

Example:

- eat(ahmad, rice).
- eat(chicken).

- Predicate is identified by its name, a slash, and its arity.

Example:

- eat(ahmad, nasi).  \rightarrow  eat/2
- eat(ahmad).        \rightarrow  eat/1
Syntax

The fundamental units of Prolog syntax are:

- Atoms,
- Numbers,
- Structures, and
- Variables

(Data objects)

- Simple objects
  - Constants
    - Atom
    - Numbers
  - Variables

- Structures

(Source: Bratko, 2001)
Syntax - Atoms

- Used as names of individuals and predicates.
- Begins with a lowercase letter
- Can contains letters, digits, and the underscore mark (_)

Example:

a
kedah
fatihah1982
muhamad_shahrul_aiman_rashid
‘Malaysia’
‘17638’
Syntax - Numbers

- Comprises of integer and real number.

Example:

<table>
<thead>
<tr>
<th>Integer</th>
<th>Real number</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1.254</td>
</tr>
<tr>
<td>12</td>
<td>0.124</td>
</tr>
<tr>
<td>0</td>
<td>0.000009</td>
</tr>
<tr>
<td>100</td>
<td></td>
</tr>
<tr>
<td>789</td>
<td></td>
</tr>
</tbody>
</table>
Syntax - Structure

- Comprises of several components – atom, bracket (open and close), and argument (inside the bracket).

- Example:

  on_top(book,table).
  country(malaysia).
  ‘State’(kedah).
  netbook(brand(acer),price(1500)).
The atom at the beginning is called the **FUNCTOR** of the structure.

If some of the arguments are also structures, the functor at the beginning of the whole thing is called the **PRINCIPAL FUNCTOR**.

```
netbook(acer, 1500).
```
Syntax - Variable

- Contains of a string of letters, digit, and underscore.
- Begin with capital letters or the underscore mark.

Example:

A
Student_Name
_College
Var1
_name
Id_17638
Syntax - Variable

- A special variable - *anonymous* variable “underscore (_) character”.

- Don't care how this variable is instantiated - don't care which term it's bound to, as long as it's bound to something.

Example:

```prolog
?- is_in(X, _).
X = sintok
X = jitra
X = kuantan
X = muar
```
Exercise

What are these and is it legal?

country(indonesia).
Book(software_engineering)
_on_table
123574_nom
eat(mahadi,rice,chicken).
‘Country’(‘Malaysia’).
5,000.04
78854
Exercise

What is wrong with this one?

capital of (alor star, kedah).
Pattern Matching

2 = A NUMBER
1 = A NUMBER

2 = 1
Pattern Matching

- Matching is a process that takes as input two terms and checks whether they match.

- The matching operator is “=“.

- Example:

  \[ a = a \]
  \[ \text{state(kedah)} = \text{state(kedah)} \]
Pattern Matching

Two objects/terms are match if:

- they are IDENTICAL, or
- the variables in both terms can be INSTANTIATED to objects.
Pattern Matching

IDENTICAL object – objects are the same.
Pattern Matching

- IDENTICAL predicates - the predicate properties must be the same, i.e;
  - The predicate name,
  - Number of argument/arity
  - The sequence/order of arguments in the predicate
Pattern Matching

Example:

\[ \text{is\_in}(\text{kedah}, \text{malaysia}) = \text{is\_in}(\text{kedah}, \text{malaysia}) \]

2 arity \rightarrow 2 arity

Same predicate name

Same sequence of arguments
Pattern Matching

Example: is_in(kedah, Malaysia).

is_in(kedah, Malaysia).

is_in(kedah, Malaysia).

is_in
kedah,
malaysia
.
Pattern Matching

- In Prolog, matching process is also called UNIFICATION.

- Unification between:
  - Query with the fact
  - Query with head of a rule
Pattern Matching

Variable in:

- One side \( \text{foo}(a) = \text{foo}(A) \).
- Both side \( \text{foo}(A) = \text{foo}(A) \).
- Mix (both contains variable & non-variable) \( \text{foo}(A,b) = \text{foo}(a,B) \). \( \text{foo}(A,b) = \text{foo}(A,B) \).
Pattern Matching

To become identical – the variables will be instantiated – INSTANTIATION.

Assign value to a variable in order to achieve a match.

Example: 

\[ \text{foo}(a) = \text{foo}(A). \]
\[ \text{foo}(A) = \text{foo}(A). \]
\[ \text{foo}(A, b) = \text{foo}(a, B). \]
\[ \text{foo}(A, b) = \text{foo}(A, B). \]

\[ A = a \]
\[ A = A \]
\[ A = a, B = b \]
\[ A = A, B = b \]
Success or fail

1. point(A,B) = point(1,2).
2. point(A,B) = point(X,Y,Z).
3. plus(2,2) = 4.
4. +(2,D) = +(E,2).
5. triangle(point(-1,0),P2,P3)=triangle(P1,point(1,0), point(0,Y)).
6. plus(2,2) = P.
7. Siti = penyanyi(siti).
8. eat = makan.
Example

binatang(comel).
binatang(tompok).
binatang(hitam).
binatang(boboy).

makan(comel, ikan).
makan(tompok, ikan).
makan(boboy, jagung).
makan(hitam, nasi).

mengiau(comel).
mengiau(tompok).
mengiau(hitam).
mengiau(boboy).

kucing(X):-
    binatang(X),
makan(X, ikan),
mengiau(X).

binatang(X) = binatang(comel)
makan(X, ikan) = makan(comel, ikan)
mengiau(X) = mengiau(comel)
Example

binatang(comel).
binatang(tompok).
binatang(hitam).
binatang(boboy).

makan(comel, ikan).
makan(tompok, ikan).
makan(boboy, jagung).
makan(hitam, nasi).

mengiau(cindai).
mengiau(tompok).
mengiau(hitam).
mengiau(boboy).

kucing(X):-
    binatang(X),
    makan(X, ikan),
    mengiau(X).

binatang(X) = binatang(comel)
makan(X, ikan) = makan(comel, ikan)
mengiau(comel) - ERROR - *Predicate not defined*
Backtracking

- Prolog will automatically backtrack – for satisfying a goal.

- Useful – relieves the programmer of the burden of programming backtracking explicitly.

- In console, Prolog will backtrack automatically after we press “;”.

Backtracking

- This is inconvenience to some problem.

Example:

?- fruit(X), write(`I like to eat `), write(X), nl.
I like to eat orange
X = orange;

I like to eat apple
X = apple;

---
Backtracking

To force backtracking use fail/0.

Example:

?- fruit(X), write(`I like to eat`), write(X), nl, fail.

I like to eat orange
I like to eat apple
I like to eat banana
Operators Notation

Data structure in Prolog:

functor(arg1, arg2, ..., argN).

Functor can become operator

example:

eat(ali,rice).

Can be written as:

ali eat rice.
## Operators Notation

### Types of operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infix</td>
<td>3 + 4</td>
</tr>
<tr>
<td>Prefix</td>
<td>+(3,4)</td>
</tr>
<tr>
<td>Postfix</td>
<td>8 factorial</td>
</tr>
</tbody>
</table>
Linguistic Operator

- Operator can be defined using \texttt{op/3}.

- \texttt{op/3} - Declares an operator with a given type and precedence.

**Syntax:**

\texttt{op(Precedence, Type, Name)}

- \texttt{Precedence}: Integer \(0-1199\)
- \texttt{Type}: Atom or list of atoms
- \texttt{Name}: Name of the operator
Linguistic Operator

- Precedence value for operator

  - The value between 0 – 1199.

  - The precedence value can influence the structure meaning:
    - low precedence meaning high priority.
    - Example: operator with 33 is given higher priority compare to 35.
    - operator with 33 will be first executed.
Linguistic Operator – Precedence Value

- The value between 0 – 1199.
- The precedence value can influence the structure meaning:
  - low precedence meaning high priority.
  - Example: operator with 33 is given higher priority compare to 35.
  - operator with 33 will be first executed.
Linguistic Operator – Precedence Value

Example

Use display/1 to view the standard syntax:

| ?- op(35,xfx,is_in).
  yes

| ?- op(33,fx,room).
  yes

| ?- display(apple is_in room kitchen).
  is_in(apple,room(kitchen))

| ?- op(36,fx,room).
  yes

| ?- display(apple is_in room kitchen).
  ! Error 42 : Syntax Error
  ! Goal : ered(_11848,_11850)

| ?- display(room kitchen is_in house).
  room(is_in(kitchen,house))

Wrongly use operator: Change the precedence value

Defining operator ‘is_in’ and ‘room’
**Linguistic Operator – Precedence Value**

**Example**

```
?- op(35,xfx,is_in).  
yes

?- op(33,fx,room).  
yes

?- op(36,fx,room).  
yes

?- display(apple is_in room kitchen).  
  is_in(apple,room(kitchen))

?-

?-

?-

?-

?- display(room kitchen is_in house).  
  room(is_in(kitchen,house))
```
Linguistic Operator – Defining Type

- If the precedence value is the same Prolog will read either from left to right or right to left based on the type.

- Type for operator

<table>
<thead>
<tr>
<th>Types</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>fx</td>
<td>non-associative prefix operator</td>
</tr>
<tr>
<td>fy</td>
<td>right associative prefix operator</td>
</tr>
<tr>
<td>xf</td>
<td>non-associative postfix operator</td>
</tr>
<tr>
<td>yf</td>
<td>left associative postfix operator</td>
</tr>
<tr>
<td>xfx</td>
<td>non-associative infix operator</td>
</tr>
<tr>
<td>xfy</td>
<td>right associative infix operator</td>
</tr>
<tr>
<td>yfx</td>
<td>left associative infix operator</td>
</tr>
</tbody>
</table>

- prefix
- postfix
- infix
Using Linguistic Operator

Example cth.pl

% Linguistic operators
:- op(800, fx, if),
   op(700, xfx, then),
   op(300, xfy, or),
   op(200, xfy, and).

% Facts
binatang(comel).
binatang(tompok).

makan(comel, ikan).
Makan(tompok, ikan).

mengiau(comel).
mengiau(tompok).

% Rules
if binatang(X) and makan(X, ikan) then kucing(X).
Using Linguistic Operator

Examine

% Rules
if binatang(X) and makan(X,ikan) then kucing(X).

if(then(and(binatang(_13214), makan(_13214, ikan)), kucing(_13214)))

Standard Syntax
2 + 1 = ?

- Arithmetic – how we do calculation in Prolog.
- Use predicate “is/2” not “=/2“.
Arithmetic Operation

- is/2 takes an arithmetic expression on its right, evaluates it, and unifies the result with its argument on the left.

- Format:

  `<argument> is <arithmetic expression>.

Example:

X is 2 + 1.
X is 9.
A = 1, X is A + 5.
2 is 1 + 1.
Arithmetic Operation

Wrong expression:

2 + 1 is 3.  \(\rightarrow\) Type error - not in the right format.

2 is 1 + X  \(\rightarrow\) Instantiation error – value of X is unknown.
Arithmetic Operation

Different with “is/2” and “=/2“:

- is/2 evaluate its right argument and unifies it with its argument on the left – do calculation.

- =/2 is to test if both arguments on the left and right are unify – do unification.
# Arithmetic Operators

## Common arithmetic operators (infix operators)

<table>
<thead>
<tr>
<th>Operator &amp; Symbol</th>
<th>Example &amp; Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>X is 3 + 2.</td>
</tr>
<tr>
<td>-</td>
<td>X is 3 - 2.</td>
</tr>
<tr>
<td>*</td>
<td>X is 3 * 2.</td>
</tr>
<tr>
<td>/</td>
<td>X is 3 / 2.</td>
</tr>
<tr>
<td>//</td>
<td>X is 3 // 2.</td>
</tr>
</tbody>
</table>
## Other arithmetic functions supported in LPA Prolog

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X^Y$</td>
<td>$X$ to the power of $Y$. e.g. $\text{abs}(-3.5)$ returns 3.5.</td>
</tr>
<tr>
<td>$\text{abs}(X)$</td>
<td>the absolute value of $X$. e.g. $\text{abs}(-3.5)$ returns 3.5.</td>
</tr>
<tr>
<td>$\text{acos}(X)$</td>
<td>the arccosine of $X$ in degrees.</td>
</tr>
<tr>
<td>$\text{aln}(X)$</td>
<td>$e$ to the power of $X$.</td>
</tr>
<tr>
<td>$\text{alog}(X)$</td>
<td>10 to the power of $X$.</td>
</tr>
<tr>
<td>$\text{asin}(X)$</td>
<td>the arcsine of $X$ in degrees.</td>
</tr>
<tr>
<td>$\text{atan}(X)$</td>
<td>the arctangent of $X$ in degrees.</td>
</tr>
<tr>
<td>$\text{cos}(X)$</td>
<td>the cosine of $X$ degrees.</td>
</tr>
<tr>
<td>$fp(X)$</td>
<td>the fractional part of $X$. e.g. $fp(-3.5)$ returns -0.5.</td>
</tr>
<tr>
<td>$\text{int}(X)$</td>
<td>the first integer equal to or less than $X$. e.g. $\text{int}(-3.5)$ returns -4.</td>
</tr>
<tr>
<td>$\text{ip}(X)$</td>
<td>the integer equal part of $X$. e.g. $\text{ip}(-3.5)$ returns -3.</td>
</tr>
<tr>
<td>$\text{ln}(X)$</td>
<td>the natural logarithm of $X$.</td>
</tr>
<tr>
<td>$\text{log}(X)$</td>
<td>the base 10 logarithm of $X$.</td>
</tr>
<tr>
<td>$\text{max}(X,Y)$</td>
<td>the maximum value of $X$ and $Y$. e.g. $\text{max}(-3.5,4)$ returns 4.</td>
</tr>
<tr>
<td>$\text{min}(X,Y)$</td>
<td>the minimum value of $X$ and $Y$. e.g. $\text{min}(-3.5,4)$ returns -3.5.</td>
</tr>
<tr>
<td>$\text{rand}(X)$</td>
<td>a random floating point number between zero and $X$.</td>
</tr>
<tr>
<td>$\text{sign}(X)$</td>
<td>-1 if $X$ is negative, 0 if $X$ is 0, or 1 if $X$ is positive. e.g. $\text{sign}(-3.5)$ returns -1.</td>
</tr>
<tr>
<td>$\text{sin}(X)$</td>
<td>the sine of $X$ degrees</td>
</tr>
<tr>
<td>$\text{sqrt}(X)$</td>
<td>the square root of $X$.</td>
</tr>
<tr>
<td>$\tan(X)$</td>
<td>the tangent of $X$ degrees.</td>
</tr>
</tbody>
</table>
Arithmetic Operators

The precedence of operators:

Example:

\[ X = 1 - 2 \times 5^2 / 8 + 6 \]

Equivalent to

\[ X = (1 - ((2 \times (5^2)) / 8) + 6) \]

Where \(^\) is performed first, then \(*\) and \(/\), and finally \(+\) and \(-\).
Using Arithmetic Operators

Example:

?- papar(5).
1
21
321
4321

1. printOut(N):-
2. printOut(1, N).
3. printOut(N, N).
4. printOut(X, N):-
5. printOut_num(X),
6. nl,
7. NewX is X + 1,
8. printOut(NewX, N).
9. printOut_num(0).
10. printOut_num(X):-
11. write(X),
12. NewX is X - 1,
13. printOut_num(NewX).
Different between Prolog an other programming languages is:

- Other programming language evaluate arithmetic expressions whenever they occur.
- Prolog evaluates them only in specific places.
Constructing Expression

List of built-in Predicates that evaluate expressions.

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>R is Ex</code></td>
<td>Evaluates Ex and unifies result with R</td>
</tr>
<tr>
<td><code>Ex1 =:= Ex2</code></td>
<td>test if the results of two expressions are equal</td>
</tr>
<tr>
<td><code>Ex1 =\= Ex2</code></td>
<td>test if the results of two expressions are not equal</td>
</tr>
<tr>
<td><code>Ex1 &gt; Ex2</code></td>
<td>test if the result of one expression is greater than another</td>
</tr>
<tr>
<td><code>Ex1 &lt;= Ex2</code></td>
<td>test if one expression is greater than or equal to another</td>
</tr>
<tr>
<td><code>Ex1 =&lt; Ex2</code></td>
<td>test if one expression is less than or equal to another</td>
</tr>
</tbody>
</table>
Exercise

Explain which of the following succeed, fail, or raise error conditions, and why:

1. ?- 5 is 2 + 3.
2. ?- 5 =:= 2+3.
3. ?- 5 = 2 + 3.
4. ?- 5 is 2 + What.
5. ?- 4+1 = 2+3.
6. ?- 4+1 is 2+3.
7. ?- 4+1 =:= 2+3.
8. ?- What is 2+3.
9. ?- What =:= 2+3.
10. ?- What is 5.
11. ?- What = 5.
12. ?- What =\= 2+3.
Practical Calculation

When using expressions in Prolog, we are mixing the styles of encoding knowledge.

Normal syntax:

? - X is 2 + 3.

Can be written as:

?- sum(2, 3, X).
Practical Calculation

Example:

doubleValue(X, Z):-
    Z is X * X.

areaOfSquare(P, L, Area):-
    Area is P * L.