STIN2024
Pengaturcaraan Logik
Logika Pemrograman
February
2nd Session 2014/2015 (A142)

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

Constructing Prolog Program

- Defining facts and rules
- Using connectors
- Constructing query
- Problem representation
- Input & output predicates
- Subroutines

STIN2024

Pengaturcaraan Logik

Logic Programming

The Eminent Management University
Defining Fact

- Facts – to describe the relationship between objects.
- To represent specific knowledge.
- Example: “Alor Setar is a capital of Kedah”

```
capital_of(alor_setar, kedah).
state(kedah, alor_setar).
is_in(alor_setar, kedah).
```
Defining Fact

**Syntax**

- Name of predicate and object must be an ATOM.
- The relation is written before the objects.
- End with “.”

relationship(arg_1, arg_2, arg_N).
Defining Rule

Rule - clause that depend on other facts.

Example:

\[\text{like}(A, B):- \]
\[\text{toy}(B), \text{play}(A, B).\]

%Facts
\[\text{toy}(\text{bear}).\]
\[\text{toy}(\text{snoopy}).\]
\[\text{toy}(\text{car}).\]
\[\text{play}(\text{ann, snoopy}).\]
\[\text{play}(\text{ann, comel}).\]
\[\text{play}(\text{ann, bear}).\]
Defining Rule

Syntax:

- `goal :-`
- `Head & body separated by "::"`
- `Head`
- `Body`
- `Head & body separated by "::"`
- `Body`
- `Subgoal separated by "," or ";"`
- `Full stop`
- `Subgoal separated by "," or ";"`
- `Full stop`
Defining Rule

Example (IF-THEN)

IF A is in B AND B is in C THEN A is in C.

In Prolog

```
is_in(A, C):-
in(A, B),
in(B, C).
```
Defining Rule

Example (IF-THEN)

IF A is clever
OR A is smart
THEN A is intelligent

intelligent(A):-
clever(A); smart(A).
Defining Rule

Example (Logical statement)

for all X and Y,
  X is the mother of Y if
  X is a parent of Y and
  X is a female.

mother(X,Y):-
  parent(X,Y),
  female(X).
Using Connector

Two or more queries or sub goals are connected by the connectors.

Three main connectors:

- **AND** ",”
- **OR** “;”
- **NOT** “\+ “ or “NOT”
Connector - AND

■ Split with ",”

■ Query:

?- in(city_plaza, alor_star),
in(alor_star, kedah).

■ Rule:

intelligent:-

clever,

smart.
Connector - OR

- Split with “;”

- Query:
  
  ?- in(city_plaza, alor_star);
  in(alor_star, kedah).

- Rule:

  intelligent:-
  clever;
  smart.
Connector - NOT

- Start with "\+" or "not"

- Query:
  
  ?- \+ in(city_plaza, alor_star).

- Rule:
  
  dumb:-

  \+ clever.
Establishing Query

Why needs query?

- To test relationships especially rules.
- To obtain knowledge from a system.
Establishing Query

- Start with “?” and follow by “—” and end with “.”.

- Example:

  ```?- like(Who, Toy).```
Establishing Query

- **List all places in the world.**
  
  `?- is_in(X, world).`

- **Malaysia is in South East.**
  
  `?- in(malaysia, south_east).`

- **City Plaza is not in perak.**
  
  `?- \+ is_in(city_plaza, perak).`

```prolog
in(city_plaza, alor_star).
in(alor_star, kedah).
in(kedah, malaysia).
in(malaysia, south_east).
in(south_east, asia).
in(asia, world).

is_in(X,Y):-
in(X,Y).

is_in(X,Y):-
in(X,T),
is_in(T,Y).
```
Establishing Query

- **Embedded Query**
  - Query that is embedded inside the program file.
  - Execute automatically during the compiling
  - Format:
    
    ```
    :- start.
    ```

*Note: Put the query at the right place.*
Representation of problem

Defining relations:

- Analyze a problem by considering possible relationships exist
- Identify possible queries
- Identify types of relationship (facts or rules)
- Create meaningful terms that can best describe the relationships between entities in the problem
- Identify arguments of relations
Representation of problem

- Simplified the problem – use table, diagram or chart.
Representation of problem

Kedah
  - Sintok
  - Alor Star

Johor
  - Muar

Perlis
  - Kangar

MALAYSIA
Representation of problem

- Identify general and specific knowledge and the relationship.

- Example:

**General knowledge**
“if A is in B, then whatever in A is in B as well”

**Specific knowledge**
“A is in B”
“C is in B”
“D is in A”

*General knowledge* – Describe an object in general.

*Specific knowledge* – Detail or specific description of an object.
Representation of problem

- Example:

**General knowledge**
“If any state is located in a country, then all cities located in that state will be in the same country”

**Specific knowledge**
“Kedah is in Malaysia”
“Kelantan is in Malaysia”
“Johor is in Malaysia”
“Sintok is in Kedah”
“Kota Bharu is in Kelantan”
“Muar is in Johor”
Representation of problem

Example:

General knowledge

is_in(City, Country):-
located(City, State),
located(State, Country).

Specific knowledge

located(kedah, malaysia).
located(kelantan, malaysia).
located(johor, malaysia).
located(sintok, kedah).
located(kota_bharu, kelantan).
located(muar, johor).
Representation of problem

How to query?

<table>
<thead>
<tr>
<th>Natural Language</th>
<th>Prolog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Is Muar is located in Johor?</td>
<td><code>- located(muar, johor).</code></td>
</tr>
<tr>
<td>Answer: true</td>
<td>yes</td>
</tr>
<tr>
<td>Is Sintok is located in Kelantan?</td>
<td><code>- located(sintok, kelantan).</code></td>
</tr>
<tr>
<td>Answer: wrong</td>
<td>no</td>
</tr>
<tr>
<td>Which state Sintok is located?</td>
<td><code>- located(sintok, X).</code></td>
</tr>
<tr>
<td>Answer: Kedah</td>
<td><code>X = kedah</code></td>
</tr>
<tr>
<td>Is Kota Bharu is in Malaysia?</td>
<td><code>- is_in(kota_bharu, malaysia).</code></td>
</tr>
<tr>
<td>Answer: yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Representation of problem

Example:

George is Michael’s father
Michael is Cathy’s father
Joanna is Cathy’s mother
Michael is Tom’s father
Joanna is Tom’s mother
Cathy is Mary’s mother
Tom is David’s father

A person is a grandfather of someone if he is a father of another person who is the father of that someone

Specific knowledge
General knowledge
Representation of problem

Example (proposed solution):

George is Michael’s father
Michael is Cathy’s father
Joanna is Cathy’s mother
Michael is Tom’s father
Joanna is Tom’s mother
Cathy is Mary’s mother
Tom is David’s father

facts:

father(george, michael).
father(michael, cathy).
mother(joanna, cathy).
father(michael, tom).
mother(joanna, tom).
mother(cathy, mary).
father(tom, david).
Representation of problem

Example (proposed solution):

A person is a grandfather of someone if he is a father of another person who is the father of that someone.

rule:

grandfather(X,Y):-
father(X, T),
father(T, Y).


Representation of problem

Example (proposed solution):

father(george, michael).
father(michael, cathy).
mother(joanna, cathy).
father(michael, tom).
mother(joanna, tom).
mother(cathy, mary).
father(tom, david).

\[
\text{specific knowledge (facts)}
\]

\[
\text{general knowledge (rule)}
\]

grandfather(X, Y) :-
father(X, T),
father(T, Y).
Querying the knowledge base

Knowledge base

father(george, michael).
father(michael, cathy).
mother(joanna, cathy).
father(michael, tom).
mother(joanna, tom).
mother(cathy, mary).
father(tom, david).

grandfather(X,Y):-
    father(X, T),
    father(T, Y).

Console

?- father(X, michael).
X=george

?- mother(cathy, Y).
Y=mary

?- grandfather(X,Y).
X = george ,
Y = cathy ;

... more press “;”
Querying the knowledge base

Examples of query

NL : Is Michael Cathy’s father?
Prolog : father(michael,cathy).

NL : Who is the father of Cathy?
Prolog : father(X,cathy).

NL : Who is the father of Cathy and mother of Cathy?
Prolog : father(X,cathy), mother(Y,cathy).

NL : Who are Michael’s children?
Prolog : father(michael,X).
Exercise

- Ann likes every toy she plays with
- Doll is a toy
- Snoopy is a toy
- Ann plays with Snoopy
- Sue likes everything Ann likes
Exercise (facts & rules)

facts:
  toy(doll).
  toy(snoopy).
  play(ann, snoopy).

rules:
  likes(ann, Y):-
      toy(Y),
      play(ann, Y).

  likes(sue, X):-
      likes(ann, Y).
## Exercise

<table>
<thead>
<tr>
<th>Applicant</th>
<th>Salary</th>
<th>Expenses</th>
<th>Loan Application status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Siti</td>
<td>2000</td>
<td>4000</td>
<td>REJECTED</td>
</tr>
<tr>
<td>Ahmad</td>
<td>1000</td>
<td>300</td>
<td>ACCEPTED</td>
</tr>
</tbody>
</table>
Exercise (facts & rules)

facts:

applicant(siti).
applicant(ahmad).
salary(siti, 2000).
salary(ahmad, 1000).
expenses(siti, 4000).
expenses(ahmad, 300).

rules:

status(X,rejected):-
    applicant(X), salary(X,Y),
    expenses(X,Z), Y =< Z.

status(X,accepted):-
    applicant(X), salary(X,Y),
    expenses(X,Z), Y > Z.
Output predicates

- To write or display and format output on console window or screen.

Commonly use predicates:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>write/1</td>
<td>write(Term).</td>
<td>write a term to the current output stream</td>
</tr>
<tr>
<td>nl/0</td>
<td>nl.</td>
<td>start a new line on the current output stream</td>
</tr>
<tr>
<td>display/1</td>
<td>display(Term)</td>
<td>write a term to the standard output stream in standard prefix notation</td>
</tr>
</tbody>
</table>
Output predicates

Examples:

?- write(`TIN2023`).
TIN2023yes

?- write(`TIN2023`), write(`Prolog`).
TIN2023Prologyes

?- write(`TIN2023`), nl, write(`Prolog`).
TIN2023
Prologyes

?- display(2+3).
+(2,3)yes
Output predicates

- **Predicate display/1**
  - Puts all functors in front of their arguments.
  - Useful for investigating the internal representation of Prolog terms.
  - Example:

Given X is 2+2, when

?- display(X is 2+2), Prolog will show

is(X,+(2,2))
Output predicates

- Limitation of write/1
  - displays quoted atoms without quotes.
  - cannot easily be read back in using Prolog syntax.
  - Example: `write(`hello there`)` will display `hello there` – without quotes.

- writeq/1
  - Display the term with quotes – can be read back in.
### Output predicates

#### Other output predicates:

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>writeq/1</td>
<td>writeq(Term).</td>
<td>write a quoted term to the current output stream</td>
</tr>
<tr>
<td>write_canonical/1</td>
<td>write_canonical(Term )</td>
<td>write a term to the current output stream in canonical form (combine effects of writeq and display)</td>
</tr>
</tbody>
</table>
Writing Formatted Output

- `fwrite/4` - formatted write of a term
- Writes a simple term `Term` to the current output stream using the Format, FieldWidth and Modifier flag.

Syntax:

`fwrite(Format, FieldWidth, Modifier, Term)`

+ `Format`  
  <atom> in the domain \{a,b,f,i,n,r,s\}.

+ `FieldWidth`  
  <integer> in the range \([-255..255]\)

+ `Modifier`  
  <integer> in the range \([-255..255]\)

+ `Term`  
  <term>
Writing Formatted Output

The allowed formats are:

<table>
<thead>
<tr>
<th>a</th>
<th>atom</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>byte list</td>
</tr>
<tr>
<td>f</td>
<td>floating point number (uses modifier)</td>
</tr>
<tr>
<td>i</td>
<td>Integer</td>
</tr>
<tr>
<td>n</td>
<td>unsigned integer</td>
</tr>
<tr>
<td>r</td>
<td>arbitrary radix (uses modifier)</td>
</tr>
<tr>
<td>s</td>
<td>string</td>
</tr>
</tbody>
</table>

Example, please refer to LPA Technical Reference pg: 111-118
Output predicates

Examples:

?- write(`TIN2023`).
TIN2023yes

?- writeq(`TIN2023`).
`TIN2023`yes

?- write(`Course `), writeq(`TIN2023`).
Course `TIN2023`yes

?- display(`2` + 3).
+(2,3)yes

?- write_canonical(`2` + 3).
+(`2`,3)yes
Output predicates - discussion

1. ?- write(abc), write(cde).
2. ?- write(abc), nl, write(cde).
3. ?- writeq(abc).
4. ?- display(abc).
5. ?- write('don''t panic').
6. ?- writeq('don''t panic').
7. ?- display('don''t panic').
8. ?- write(Abc).
9. ?- writeq(Abc).
10. ?- display(Abc).
11. ?- write(2+2).
12. ?- display(2+2).
Input of terms

- To get input from user or input streams.

- Built-in predicate `read/1`

- Syntax:

  ```prolog
  read(Term).
  ```

Example:

```
| ?- read(X).
| : stin2023.
X = stin2023

| ?- read(X).
| : `STIN2023 Prolog`.
X = `STIN2023 Prolog`

| ?- read(X).
| : stin2023 prolog.
* Syntax Error
```
Input of terms

The input terms must be typed in the same syntax as if it were within a Prolog program.

Must be followed by a period.

More examples:

1. `?- read(X).`  
   `|: abc. X = abc`

2. `?- read(hussain).`  
   `|: hussain. Yes`

3. `?- read(X).`  
   `|: Y. X = _`

4. `?- read(X).`  
   `|: abc`  
   `|: a`  
   `X = abc`

5. `?- read(X).`  
   `|: a`  
   `b. * Syntax Error`
Input of terms – Usage Example

% Facts

capital_of(bandar_melaka,melaka).
capital_of(johor_baharu,johor).
capital_of(kuantan,pahang).
capital_of(kuala_terengganu,terengganu).
capital_of(kota_baharu,kelantan).
capital_of(kuching,sarawak).
capital_of(kota_kinabalu,sabah).

% Rule

go:-
write('Enter the state name'),
nl,
read(State),
capital_of(City,State),
write('Its capital is: '),
write(City),
nl.
Input of terms – Usage Example

Query and output example

?- go.

Enter the state name
|: kelantan.

Its capital is: kota_baharu
Reading Formatted Data

- fread/4 - formatted read of a term
- Read a simple term Term from the current input stream using the Format, FieldWidth and Modifier flag.
- Syntax:

  \[ \text{fread(Format, FieldWidth, Modifier, Term)} \]

  +Format <atom> in the domain \{a,b,f,i,n,r,s\}.
  +FieldWidth <integer> in the range [-255..255]
  +Modifier <integer> in the range [-255..255]
  -Term <variable>
Reading Formatted Data

- The allowed formats are:

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>atom (uses modifier)</td>
</tr>
<tr>
<td>b</td>
<td>byte list (uses modifier)</td>
</tr>
<tr>
<td>f</td>
<td>floating point number (uses modifier)</td>
</tr>
<tr>
<td>i</td>
<td>Integer</td>
</tr>
<tr>
<td>n</td>
<td>unsigned integer</td>
</tr>
<tr>
<td>r</td>
<td>arbitrary radix (uses modifier)</td>
</tr>
<tr>
<td>s</td>
<td>string (uses modifier)</td>
</tr>
</tbody>
</table>

Example please refer to LPA Technical Reference pg: 102-109
### Other Character Input/Output

#### Get/read character

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get/1</td>
<td>get(N)</td>
<td>Reads the next non-white space character from the current input stream, and unifies N with the ASCII value of this character.</td>
</tr>
<tr>
<td></td>
<td>(N is variable or char)</td>
<td></td>
</tr>
<tr>
<td>get0/1</td>
<td>get0(N)</td>
<td>Reads a character from the current input stream, and unifies N with the ASCII value of this character. When the input file pointer is at the end of a file this get0/1 returns the value -1.</td>
</tr>
<tr>
<td></td>
<td>(N is variable or char)</td>
<td></td>
</tr>
<tr>
<td>getb/1</td>
<td>getb(Byte)</td>
<td>Input a byte from the keyboard or mouse. Mouse keys return -1, -2 and -3 for the pressing of the left, right and both buttons respectively.</td>
</tr>
<tr>
<td></td>
<td>(Byte is a variable)</td>
<td></td>
</tr>
</tbody>
</table>
## Other Character Input/Output

### Display/print character

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>put/1</td>
<td>put(N)</td>
<td>Writes the character whose ASCII code is N to the current output stream. N can be an integer in the ASCII range (0 to 255), or an expression that evaluates to an integer in the ASCII range.</td>
</tr>
<tr>
<td></td>
<td>(N is char)</td>
<td></td>
</tr>
<tr>
<td>putb/1</td>
<td>putb(Byte)</td>
<td>Output to the screen the ASCII character related to the ASCII value Byte. If Byte is a negative integer then two characters are output to the console window: the first is the null character (0), followed by the character related to the absolute value of Byte.</td>
</tr>
<tr>
<td></td>
<td>(Byte is char)</td>
<td></td>
</tr>
</tbody>
</table>
Computing Vs. Printing

- Using output predicates such as write, and display will force Prolog to print the result or output on screen.

example:

?- like(ann, X), write(X).

- will force Prolog to look for what ann like and print it on the screen.
Computing Vs. Printing

- In contrast

  ?- like(ann, X).

- will force Prolog to look for what Ann like but no output is force to be printed on the screen.

- By default Prolog will print the value of X which is instantiated during the matching process.

- Maintaining the value of X is beneficial when passing a value to other subgoal in the same program.
Computing Vs. Printing

In other programming language, passing or returning the value is done as follows:

Example:

```c
... 
Z = add(X,Y);
...
Z2 = mul(X,Y);
...

int add(int X, int Y)
{
    return X + Y;
}

int mul(int X, int Y)
{
    return X * Y;
}
```
Computing Vs. Printing

Exercise

```prolog
cal(X, Y, Z):-
  add(X, Y, Z),
  write(Z),
  mul(X, Y, Z2),
  write(Z2).

add(X, Y, Z):-
  Z is X + Y.

mul(X, Y, Z):-
  Z is X * Y.
```

What is the output?
Predicates Vs. Subroutines

- The rule defines a subroutine – all subgoals can be execute through single query.

- Writing all subgoals in one rule in inefficient in Prolog.

Example:

```prolog
print_veg:-
    veg(X),
    write(`I like to eat vegetable `),
    write(X), nl,
    fail.
```
Predicates Vs. Subroutines

Split the program into separate operations.

Eg: printing the vegetables in the desired format and backtracking through all alternatives

Example

```prolog
print_veg:-
    veg(X),
    write('I like to eat vegetable '),
    write(X), nl.

print_vegs:-
    print_veg, fail.
```
Term and Case Conversion

atom_chars/2 - converts between an atom and a list of characters

atom_chars(Atom, CharList )

Atom  <variable> or <atom>

CharList  <char_list> or <variable>
Term and Case Conversion

atom_chars/2 – example:

?- atom_chars(eat, CharList ).
CharList = [101,97,116]

Atom = eat
Term and Case Conversion

atom_string/2 - convert between an atom and a string

atom_string(Atom, String)

Atom  <atom> or <variable>

String <string> or <variable>
Term and Case Conversion

atom_string/2 – example:

?- atom_string( eat, String).
String = `eat`

?- atom_string( Atom, `eat`).
Atom = eat
Term and Case Conversion

- `number_atom/2` - convert between a number and an atom

`number_atom(Number, Atom)`

- `Number` <number> or <variable>
- `Atom` <atom> or <variable>
Term and Case Conversion

\textbf{number\_atom/2} – example:

\begin{verbatim}
?- number\_atom(123, Atom).
Atom = '123'

?- number\_atom(Number, '123').
Number = 123
\end{verbatim}
Term and Case Conversion

- `number_chars/2` - convert between numbers and a list of characters

```
number_chars(Number, CharList )
```

Number: `<number>` or `<variable>`
CharList: `<char_list>` or `<variable>`
Term and Case Conversion

- `number_chars/2` – example:

```
?- number_chars(123, CharList ).
CharList = [49,50,51]

?- number_chars(Number, [49,50,51] ).
Number = 123
```
Term and Case Conversion

- number_string/2 - convert between a number and a string

number_string(Number, String )

- Number  <number> or <variable>
- String   <string> or <variable>
Term and Case Conversion

- `number_string/2` – example:

  ?- number_string(123, String ).
  String = `123`

  ?- number_string(Number, `123` ).
  Number = 123
Term and Case Conversion

- `string_chars/2` - convert between strings and character lists

```
string_chars(String, CharList)
```

- `String` - `<string>` or `<variable>`
- `CharList` - `<char_list>` or `<variable>`
Term and Case Conversion

\textbf{string_chars/2} – example:

? - string_chars( `eat`, CharList).  
CharList = [101,97,116]

? - string_chars( String, [101,97,116]).  
String = `eat`
Term and Case Conversion

- `lwrupr/2` - convert between lower and upper case

\[ lwrupr(Lower,Upper) \]

- Lower: <atom>, <string> or <variable>
- Upper: <atom>, <string> or <variable>
Term and Case Conversion

\texttt{lwrupr/2} – example:

\begin{verbatim}
?- lwrupr(eat,Upper).
Upper = 'EAT'

?- lwrupr(Lower,'EAT').
Lower = eat
\end{verbatim}
Term and Case Conversion

=../2 - "univ": define the relationship between a term and a list

Term =.. List

Term <term> or <variable>
List <list> or <variable>
Term and Case Conversion

=../2 – example:

?- eat(ahmad,rice) =.. U.
U = [eat,ahmad,rice]

?- P =.. [eat,ahmad,rice].
P = eat(ahmad,rice)