Clauses and Predicates

TK2ICM: Logic Programming (2nd Term 2018-2019)

M. Arzaki

Computing Laboratory, School of Computing
Telkom University

SoC Tel-U

January-February 2019

Acknowledgements

This slide is compiled using the materials in the following sources:
Books:

1. P. Blackburn, J. Bos, K. Striegnitz, *Learn Prolog Now!* (Chapter 1-6, 10,11), London: College Publications (available online), 2006. **[LPN]**
2. M. Bramer, *Logic Programming with Prolog* (Chapter 1-9), 2nd Edition, Springer, 2013. **[LPwP]**
3. I. Bratko, *Prolog Programming for Artificial Intelligence* (Chapter 1-3, 5,6,8,9), Pearson Education, 2001. (advanced reference). **[PPAI]**
4. K. H. Rosen, *Discrete Mathematics and Its Applications* (Chapter1), 7th Edition, 2012.
5. M. Ben-Ari, *Mathematical Logic for Computer Science* (Logic Programming Sections), 2nd Edition, 2000.

Lecture slides and lecture notes:

1. *Prolog Programming* by Kristina Striegnitz.
2. *Learn Prolog Now!* by Patrick Blackburn, Johan Bos, and Kristina Striegnitz.
3. *Logic Programming* at Fasilkom UI by A. A. Krisnadhi and A. Saptawijaya.
4. *Computational Logic Part 2: Logic Programming* at Fasilkom UI by L. Y. Stefanus.
5. *Logic Programming* at ILLC, University of Amsterdam by U. Endriss.
6. *Functional Programming* at Fasilkom UI by A. Azurat.
7. *Bahasa Prolog* at FPMIPA UPI by Munir.
8. Other available sources online.

Contents

Contents

Clauses

- A clause can run over more than one line or there may be several on the same line.
- **A clause is terminated by a dot character**, followed by at least one ‘white space’ character, e.g., a space or a carriage return.
- There are two types of clauses: *facts* and *rules*.
- Facts are of the form: `head.`

Some examples of facts:

```
christmas.
likes(john,mary).
likes(X,prolog).
dog(fido).
```

In the above Prolog script, `christmas` is a predicate (or functor) of arity 0.

Rules

Rules are of the form:

```
<head>:- t1,t2,...,tn
```

where n ≥ 1

- <head> is called the head of the clause (or the head of the rule) and, as for facts, must be a call term.
- `:-` is called the neck of the clause (or the 'neck operator'). It is read as 'if'.
- `t1,t2,...,tn` is called the body of the clause (or the body of the rule), It specifies the conditions that must be met in order for the conclusion, represented by the head, to be satisfied.

Some examples of rules:

```
large_animal(X):- animal(X),large(X).
grandparent(X,Y):- father(X,Z),parent(Z,Y).
go:- write('hello world'),nl.
```

Finding Large Animals

```
dog(fido).  dog(rover).  dog(jane).
dog(tom).  dog(fred).  dog(henry).
cat(mary).  cat(harry).  cat(bill).  cat(steve).
large(fido).  large(mary).  large(tom).
large(fred).  large(steve).  large(jim).  large(mike).

large_animal(X):- dog(X), large(X).
large_animal(Y):- cat(Y), large(Y).
```

Test the above program. Do you see something peculiar? Can you improve it?

Contents

Predicates

The following simple program has **five clauses**. For each of the first three clauses,the head is a compound term with functor parent and arity 2 (i.e., two arguments).

```
parent(victoria,albert).
parent(X,Y):- father(X,Y).
parent(X,Y):- mother(X,Y).
father(john,henry).
mother(jane,henry).
```

It is possible for the program also to include clauses for which the head has functor `parent`, but a different arity, for example

```
parent(john).
parent(X,Y):- son(X,Y).
/* X is a parent of Y if X has a son Y*/
```

Note that a query such as

```
?- listing(mypred).
```

gives a listing of all the clauses for predicate mypred whatever the arity.
Question: what is the result of the query ?- listing(parent) in the following script?

```
parent(john).
parent(X,Y):- son(X,Y).
/* X is a parent of Y if X has a son Y*/
```

Contents

Declarative and Procedural Interpretations of Rules

Rules have both a declarative and a procedural interpretation.

For example, the declarative interpretation of the rule

```
chases(X,Y):-    dog(X),cat(Y),write(X),
                              write(' chases '),write(Y),nl.
```

is: `chases(X,Y)` is true if `dog(X)` is true and `cat(Y)` is true and `write(X)` is true and `write(' chases ')` is true and `write(Y)` is true and `nl` is true.

The procedural interpretation of the above rule is: to satisfy `chases(X,Y)`, satisfy the following condition sequentially:

1. `dog(X)`,
2. `cat(X)`,
3. `write(X)`,
4. `write(' chases ')`,
5. `write(Y)`,
6. `nl`.

Notes on Declarative and Procedural Interpretations

- The order of the clauses defining a predicate and the order of the goals in the body of each rule are irrelevant to the declarative interpretation.
- However, this order is of vital importance to the procedural interpretation.
- When evaluating a goal, the clauses in the database are examined from top to bottom.
- Where necessary, the goals in the body of a rule are examined from left to right.

Contents

Some Built-in Predicates

- There are standard predicates predefined by the Prolog system.
- These are known as built-in predicates (BIPs) and **may not be redefined by a user program**.
- Some examples are: `write/1`, `nl/0`, `repeat/0`, `member/2`, `append/3`, `consult/1`, `halt/0`.
- Some BIPs are common to all versions of Prolog. Others are version-dependent.
- Two of the most commonly used built-in predicates are `write/1` and `nl/0`.

The `write/1` Predicate

The `write/1` predicate takes a term as its argument, e.g.:

```
write(hello)
write(X)
write('hello world')
```

- Providing its argument is a valid term, the `write` predicate **always succeeds** and as a side effect writes the value of the term to the user's screen.
- To be more precise it is output to the current output stream, which by default will be assumed to be the user's screen.
- If the argument is a quoted atom, e.g. 'hello world', the quotes are not outputted.

The `nl/0` Predicate

- The `nl/0` predicate is an atom, i.e. a predicate that takes no arguments.
- The predicate **always succeeds** and as a side effect starts a new line on the user's screen.

Contents

Simplifying Entry Goals

In developing or testing programs it can be tedious to enter repeatedly at the system prompt a lengthy sequence of goals such as

```
?- dog(X),large(X),write(X),write(' is a large dog'),nl.
```

A commonly used programming technique is to define a predicate such as go/0 or start/0, with the above sequence of goals as the right-hand side of a rule, e.g.,

```
go:- dog(X),large(X),write(X),
           write(' is a large dog'),nl.
```

Recursion

- An important technique for defining predicates, which will be used frequently later in this course, is to define them in terms of themselves.
- This is known as a recursive definition. There are two forms of recursion:
  1. Direct recursion. Predicate `pred1` is defined in terms of itself.
  2. Indirect recursion. Predicate `pred1` is defined using `pred2`, which is defined using `pred3`, . . . , which is defined using `pred1`.
- The first form is more common.

Example

Suppose John likes anyone who likes a dog (the person likes at least one dog). We can express this as

Recursion

- An important technique for defining predicates, which will be used frequently later in this course, is to define them in terms of themselves.
- This is known as a recursive definition. There are two forms of recursion:
  1. Direct recursion. Predicate `pred1` is defined in terms of itself.
  2. Indirect recursion. Predicate `pred1` is defined using `pred2`, which is defined using `pred3`, . . . , which is defined using `pred1`.
- The first form is more common.

Example

Suppose John likes anyone who likes a dog (the person likes at least one dog). We can express this as

```
likes(john,X):- likes(X,Y),dog(Y).
```

Predicates and Functions

- The use of the term ‘predicate’ in Prolog is closely related to its use in mathematical logic.
- A predicate can be thought of as a relationship between a number of values (its arguments) such as `likes(henry,mary)` or `X = Y`, which can be either true or false.
- This contrasts with a function, such as $6 + 4$, the square root of 64, or the first three characters of ‘hello world’, which can evaluate to a number, a string of characters or some other value as well as true and false.
- Prolog does not make use of functions except in arithmetic expressions.

Contents

The consult/1 Predicate

- We use the consult/1 predicate to load the clauses contained in a text file.
- A Prolog program is just a collection of clauses (rules and facts) so we will refer to a file used this way as a program file.

Suppose that file testfile.pl contains the following lines:

```
alpha.
beta.

dog(fido).
dog(misty).
dog(harry).

cat(jane).
cat(mary).
```

then the query

```
?- consult('testfile.pl').
```

puts all seven of the above clauses into the database.

Directory Syntax

Remark

Filenames are considered to be relative to the directory from which Prolog started up. To consult a file in another directory, the usual file naming conventions are available, e.g. `?- consult('/mydir/testfile.pl').` Notice that we use the regular slash instead of the backslash to define the directory.

The invocation

```
?- consult('mydir/testfile.pl').
```

is equivalent to the following syntax

```
['mydir/testfile.pl'].
```

A Note for Windows User

To consult a file, we can simply do the following steps:

1. Click File.
2. Choose Consult . . .
3. Browse the `.pl` file we want to consult.

Consulting Two or More Files

Suppose we have the file `testfile1.pl` as follows:

```
alpha.
beta.

dog(fido).
dog(misty).
dog(harry).

cat(jane).
cat(mary).
```

In addition, we have the file `testfile2.pl` as follows:

```
gamma.

dog(patch).

elephant(dumbo).
elephant(fred).
```

The query:

```
?- consult('testfile1.pl'), consult('testfile2.pl').
```

places the following clauses in the database:

The query:

```
?- consult('testfile1.pl'), consult('testfile2.pl').
```

places the following clauses in the database:

```
gamma.
alpha.
beta.

dog(patch).
cat(jane).

cat(mary).

elephant(dumbo).
elephant(fred).
```

Some Remarks

- The atom gamma from testfile2.pl has been added to the database to join the two atoms already there from testfile1.pl.
- The two clauses for cat/1 loaded from testfile1.pl remain in the database.
- The two clauses for elephant/1 have been loaded into the database from testfile2.pl.
- Three dog/1 clauses in testfile1.pl have been deleted and replaced by the single dog/1 clause in testfile2.pl.

Important Remarks

1.

```
?- consult('testfile1.pl'), consult('testfile2.pl').
```

   and

```
?- consult('testfile2.pl'), consult('testfile1.pl').
```

   are different.

2. To consult multiple files, for example, three files, we can use:

```
?- ['myfilea.pl', 'myfileb.pl', 'myfilec.pl'].
```

   which has the same effect as

```
?- consult('myfilea.pl'),
consult('myfileb.pl'),consult('myfilec.pl').
```

Contents

Variables

- Variables can be used in the head or body of a clause and in goals entered at the system prompt.
- However, their interpretation depends on where they are used.
- Variables in goals can be interpreted as meaning ‘find values of the variables that make the goal satisfied’.
- For example, the goal `?- large_animal(A).` can be read as find a value A such that `large_animal(A)` is true.

Universally Quantified Variables

- If a variable appears in the head of a rule or fact it is taken to indicate that the rule or fact applies for all possible values of the variable.
- For example, the rule:
  `large_animal(X):- dog(X),large(X).`
  can be read as:
  'for all values of `X`, `X` is large animal if `X` is a dog and `X` is large.
- In predicate logic, we may write:
  $\forall X \left(large\_animal\left(X\right) \leftarrow dog\left(X\right) \wedge large\left(X\right)\right)$.

Existentially Quantified Variables

- If a variable, say `Y`, appears in the body of a clause but not in its head it is taken to mean 'there is (or there exists) at least one value of `Y`'.
- In the above case, `Y` is existentially quantified variable.
- For example, the rule:
  `dogowner(X):- dog(Y),owns(X,Y).`
  can be interpreted as:
  'for all values of `X`, `X` is a dog owner if there is some `X` such that `Y` is a dog and `X` owns `Y`'
- In predicate logic, we may write:
  $\forall X\ (dogowner\ (X) \leftarrow \exists Y\ (dog\ (Y) \wedge owns\ (X, Y)))$.

Anonymous Variables

Suppose now that the database contains the following clauses:

```
person(frances,wilson,female,28,architect).
person(fred,jones,male,62,doctor).
person(paul,smith,male,45,plumber).
person(martin,williams,male,23,chemist).
person(mary,jones,female,24,programmer).
person(martin,johnson,male,47,solicitor).
```

The six clauses above (all facts) comprise the definition of predicate `person/5`, which has five arguments with obvious interpretations, i.e. the forename, surname, sex, age and occupation of the person represented by the corresponding fact.

- In order to find whether there is a clause corresponding to anyone called `paul` in the database, it is only necessary to enter a goal such as:
  ```
  ?- person(paul,Surname,Sex,Age,Occupation).
  ```
- Prolog replies:
  ```
  Surname = smith ,
  Sex = male ,
  Age = 45 ,
  Occupation = plumber
  ```
- If it is only important to establish whether there is someone with forename `paul` in the database an easier way is to use the goal:
  ```
  ?- person(paul,_,_,_,_).
  ```
- Prolog replies with `true`.

Remark

The underscore character _ denotes a special variable, called the *anonymous variable*. This is used when the user does not care about the value of the variable.

Contents

Exercise 1

Suppose we have the following program

```
/* Animals Database */
animal(mammal,tiger,carnivore,stripes).
animal(mammal,hyena,carnivore,ugly).
animal(mammal,lion,carnivore,mane).
animal(mammal,zebra,herbivore,stripes).
animal(bird,eagle,carnivore,large).
animal(bird,sparrow,scavenger,small).
animal(reptile,snake,carnivore,long).
animal(reptile,lizard,scavenger,small).
```

Devise and test the following goals:

1. `mammal(X)` for finding all the mammals
2. `mammal_carnivore(X)` for finding all the carnivores that are mammals,
3. `mammal_stripes(X)` for finding all the mammals with stripes,
4. `reptile_mane(X)` for finding all the reptiles with mane.

Exercise 2

Suppose we have the following program:

```
/* Dating Agency Database */
person(bill,male).
person(george,male).
person(alfred,male).
person(carol,female).
person(margaret,female).
person(jane,female).
```

Construct a predicate `couple(X,Y)` that is `true` whenever `X` and `Y` have two different genders.