Prolog Fundamentals

Artificial Intelligence Programming in Prolog

Lecture 2

Anatomy of a Program

- A Prolog programs are made up of facts and rules.
- A fact asserts some property of an object, or relation between two or more objects.

  e.g. `parent(jane,alan).`

  Can be read as "Jane is the parent of Alan."

- Rules allow us to infer that a property or relationship holds based on preconditions.

  e.g. `parent(X,Y) :- mother(X,Y).`

  = "Person X is the parent of person Y **if** X is Y's mother."

Predicate Definitions

- Both facts and rules are predicate definitions.
- *'Predicate'* is the name given to the word occurring before the bracket in a fact or rule:

```
parent(jane,alan).
```

Predicate name

- By defining a predicate you are specifying which information needs to be known for the property denoted by the predicate to be true.

Clauses

- Predicate definitions consist of *clauses*.

  = An individual definition (whether it be a fact or rule).

  e.g. `mother(jane,alan).` = Fact

  `parent(P1,P2):- mother(P1,P2).` = Rule

  head **body**

- A clause consists of a *head* and sometimes a *body.*
- facts don't have a body because they are always true.

Arguments

- A predicate head consists of a *predicate name* and sometimes some *arguments* contained within brackets and separated by commas.

```
mother(jane,alan).
```

Predicate name **Arguments**

- A body can be made up of any number of subgoals (calls to other predicates) and terms.
- Arguments also consist of terms, which can be:
  - Constants e.g. jane,
  - Variables e.g. Person1, or
  - Compound terms (explained in later lectures).

Terms: Constants

Constants can either be:

- Numbers:
  - integers are the usual form (e.g. 1, 0, -1, etc), but
  - floating-point numbers can also be used (e.g. 3.0E7)
- Atoms or Symbolic (non-numeric) constants:
  - always start with a lower case alphabetic character and contain any mixture of letters, digits, and underscores (but no spaces, punctuation, or an initial capital).
    - e.g. abc, big_long_constant, x4_3t).
- String constants:
  - are anything between single quotes e.g. 'Like this'.

Terms: Variables

- Variables always start with an upper case alphabetic character or an underscore.
- Other than the first character they can be made up of any mixture of letters, digits, and underscores.

  e.g. `X, ABC, _89two5, _very_long_variable`

- There are no "types" for variables (or constants) – a variable can take any value.
- All Prolog variables have a "local" scope:
  - they only keep the same value within a clause; the same variable used outside of a clause does not inherit the value (this would be a "global" scope).

Naming tips

- Use real English when naming predicates, constants, and variables.

  e.g. "John wants to help Somebody."

  Could be: **`wants(john,to_help,Somebody).`**

  Not: **`x87g(j,_789).`**

- Use a Verb Subject Object structure:

  `wants(john,to_help).`

- **BUT** do not assume Prolog Understands the meaning of your chosen names!
  - You create meaning by specifying the body (i.e. preconditions) of a clause.

Using predicate definitions

- Command line programming is tedious

  e.g.
```
| ?- write('What is your name?'), nl, read(X),
        write('Hello '), write(X).
```

- We can define predicates to automate commands:

```
greetings:-
    write('What is your name?'),
    nl,
    read(X),
    write('Hello '),
    write(X).
```

Prolog Code

```
| ?- greetings.
What is your name?
|: tim.
Hello tim
X = tim ?
yes
```

Terminal

Arity

- `greetings` is a predicate with no arguments.
- The number of arguments a predicate has is called its arity.
  - The arity of `greetings` is zero = greetings/0
- The behaviour of predicates can be made more specific by including more arguments.
  - `greetings(hamish)` = greetings/1
- The predicate can then behave differently depending on the arguments passed to it.

Using multiple clauses

- Different clauses can be used to deal with different arguments.

```
greet(hamish):-
      write('How are you doin, pal?').
greet(amelia):-
      write('Awfully nice to see you!').
```

= "Greet Hamish or Amelia" = a disjunction of goals.

```
| ?- greet(hamish).          | ?- greet(amelia).
How are you doin, pal?       Awfully nice to see you!
yes                          yes
```

- Clauses are tried in order from the top of the file.
- The first clause to match succeeds (= yes).

Variables in Questions

- We can call `greet/1` with a variable in the question.
- A variable will match any head of `greet/1`.

```
| ?- greet(Anybody).
How are you doin, pal?
Anybody = hamish?
yes
```

- The question first matches the clause closest to the top of the file.
- The variable is instantiated (i.e. bound) to the value '`hamish`'.
- As the variable was in the question it is passed back to the terminal (`Anybody = hamish?`).

Re-trying Goals

- When a question is asked with a variable as an argument (e.g. `greet(Anybody).`) we can ask the Prolog interpreter for multiple answers using: **;**

```
| ?- greet(Anybody).
  How are you doin, pal?
  Anybody = hamish? ;     ← “Redo that match.”
  Anybody = amelia? ;     ← “Redo that match.”
  no                       ← “Fail as no more matches.”
```

- This fails the last clause used and searches down the program for another that matches.
  - `RETURN` = accept the match
  - `;` = reject that match

Variable clause head.

- If `greet/1` is called with a <u>constant</u> other than `hamish` or `amelia` it will fail (return `no`).
- We can create a default case that always succeeds by writing a clause with a variable as the head argument.

```
greet(Anybody):-
      write('Hullo '),
      write(Anybody).
```

```
|?- greet(bob).
Hullo bob.
yes
```

- Any call to `greet/1` will unify (i.e. match) `greet(Anybody).`
- Once the terms unify the variable is instantiated to the value of the argument (e.g. `bob`).

Ordering of clauses

- The order of multiple clauses is important.

```
greet(Anybody):-
        write('Hullo '), write(Anybody).

greet(hamish):-
  write('How are you doin, pal?').

greet(amelia):-
  write('Awfully nice to see you!').
```

```
| ?- greet(hamish).
Hullo hamish?
yes
```

- The most specific clause should always be at the top.
- General clauses (containing variables) at the bottom.

Ordering of clauses

- The order of multiple clauses is important.

```
| ?- greet(hamish).
How are you doin,
pal?.
yes
```

```
greet(hamish):-
   write('How are you doin, pal?').

greet(amelia):-
   write('Awfully nice to see you!').

greet(Anybody):-
        write('Hullo '), write(Anybody).
```

- The most specific clause should always be at the top.
- General clauses (containing variables) at the bottom.

Unification

- When two terms match we say that they unify.
  - Their structures and arguments are compatible.
- This can be checked using =/2

```
|?- loves(john,X) = loves(Y,mary).
X = mary,      ← unification leads to instantiation
Y = john?      ←
yes
```

Terms that don't unify
fred = jim.
'Hey you' = 'Hey me'.
frou(frou) = f(frou).
foo(bar) = foo(bar,bar).
foo(N,N) = foo(bar,rab).

Terms that unify	Outcome
fred = fred.	yes.
'Hey you' = 'Hey you'.	yes
fred=X.	X=fred.
X=Y.	Y = X.
foo(X) = foo(bar).	X=bar.
foo(N,N) = foo(bar,X).	N=bar, X=bar.
foo(foo(bar)) = foo(X)	X = foo(bar)

Asking questions of the database

We can ask about facts directly:

```
|?- mother(X,alan).
X = jane?
Yes
```

Or we can define rules that prove if a property or relationship holds given the facts currently in the database.

```
|?- parent(jane,X).
X = alan?
yes
```

```
mother(jane,alan).
father(john,alan).

parent(Mum,Child):-
 mother(Mum,Child).

parent(Dad,Child):-
 father(Dad,Child).
```

Summary

- A Prolog program consists of predicate definitions.
- A predicate denotes a property or relationship between objects.
- Definitions consist of clauses.
- A clause has a head and a body (Rule) or just a head (Fact).
- A head consists of a predicate name and arguments.
- A clause body consists of a conjunction of terms.
- Terms can be constants, variables, or compound terms.
- We can set our program goals by typing a command that unifies with a clause head.
- A goal unifies with clause heads in order (top down).
- Unification leads to the instantiation of variables to values.
- If any variables in the initial goal become instantiated this is reported back to the user.