Logic Programming Language

Overview

- Let's take a look at an example first.
  - When you swipe Facebook or LinkedIn, there are always some suggestions regarding who you may know. Some may be your classmates, some may be from the same institute as you, and there are always some you have no idea who they are.
  - How could these social media find possible connections for you? Does anyone have thought about this?
  - Suppose we have a small social network. The nodes at the ends of each purple edge mean they are friends on Facebook. Now, we want to find possible connections. Here we define the connection to be friend's friend.

![Find connections in this social network](image)

- In this graph, Alice and Bob are friends, Bob and Carol are friends, so Alice and Carol have a connection. So as for Bob and Mary, Carol and James, Mary and Alice, James and Bob.
- Question: If you are going to implement an algorithm to find all possible connections in a social network, how would you do that?
- Show connection.pl, go through the code, run it, and show the output.

```prolog
/* Social media connections
* Lanuch terminal
* $ swipl
* consult('connection.pl').
* friend(alice, X).
* connection(X, Y).
*/
friend(alice, bob).
friend(bob, carol).
friend(carol, mary).
friend(mary, james).
friend(james, alice).

collection(X, Y) :- friend(X, Z), friend(Z, Y).
```
Today, we are going to talk about logic programming.

Logic programming emerged as a distinct paradigm in the 1970’s.

What makes logic programming distinct and cool is that when you programming with logic program languages, you only need to say what you want, not how you want it done.

A program written in a logic programming language consists of:
- a set of facts about objects and their relationships (friends in the connection.pl)
- a set of rules about objects and their relationships (connections in the connection.pl)
- a set of queries about objects and their relationship (connection(X, Y). on the terminal)

Horn Clauses

- The rule, connection, in the above program is an example of the Horn clauses.
  connection(X, Y) :- friend(X, Z), friend(Z, Y).
- Horn clauses form the basis of logic programming.
- **Horn clauses** are named for the logician Alfred Horn, who first pointed out their significance in 1951.
- A Horn clause is a **statement** that connects a single predicate to a set of conditions for that predicate to apply.
- It has been used in logical programming, formal specification, and model theory.
- The way to read the clause is that the connection predicate applies to X and Y if X and Z are friends, Z and Y are friends.
- The predicate on the left applies to the variable X and Y only if all of the predicates on the right also apply to X, Y, Z.

Resolution

- Resolution give us the ability to start searching a database of facts by connected together clauses.
- As resolution proceeds, it finds values and bind to variables. (e.g. friend(alice, X), bind bob to X).
- If the head of a Horn clause (h) matches one of the terms of another Horn clause, then that term can be replaced by h’s terms.
  isMammal(x) :- cat(x)
  isAnimal(x) :- isMammal(x)
  isAnimal(x) :- cat(x)
- In this example, isMammal(x) is the head of the first Horn clause. It matches the term of the second Horn clause, so that the term in the second clause can be replaced by the term of the first clause.

Prolog

- **Programming with Logic**
- Very different from other programming languages
  - Declarative (not procedural)
  - Recursion (no “for” or “while” loop)
  - Relations (no function)
  - Unification
• The process of identifying the set of values for a set of variable in a Horn clause that make it true.

• Prolog and Logic

<table>
<thead>
<tr>
<th></th>
<th>Prolog</th>
<th>Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implication</td>
<td>A :- B</td>
<td>B → A</td>
</tr>
<tr>
<td>Conjunction</td>
<td>A, B</td>
<td>A \ A</td>
</tr>
<tr>
<td>Disjunction</td>
<td>A, B</td>
<td>A \ B</td>
</tr>
<tr>
<td>Not</td>
<td>not(A)</td>
<td>~A</td>
</tr>
</tbody>
</table>

• Fact
- Consist of a particular item (e.g. sunny.) or a relation between items
- Should begin with a lowercase letter and end with a full stop
- Can consist of any letter or number combination, as well as the underscore character
- Should avoid the characters +, -, *. /, or other mathematical operators
- Show facts.pl.

    tomorrow_is_another_day.
    saturday.
    no_courses.
    happy.

- Query:
  • saturday.
  • Saturday // uninitialized variable
  • friday

• Relation
- more complicate facts
- To define a relation, start with the relation name followed by a pair of parentheses with arguments in it.
  
  relation(<argument1>, <argument2>, ... , <argumentN>).
- Relation names must start with a lowercase letter.
- Can have N arguments, N >= 0.

• Variable
- A string of uppercase letters, lowercase letters, digits and underscore that start either with an uppercase letter or with an underscore.
- Can be used in queries and rules

    connection(X, Y) :- friend(X, Z), friend(Z, Y).

- X and Y are variables
• Rule
  - Rules make conditional statements.
  - Each rule can have several variations to specify alternative ways of proving a particular thing.
  - Prolog starts from the first rule/fact. If it does not succeed, Prolog tries the second. The query fails if we run out of rules/facts.
  - The variables with the same name in a rule have the same instantiation (binding to the same value) for each solution to a particular query.
  - Identical variable names in separate rules are independent.
  - Show rule.pl

```prolog
    mammal(X) :- cat(X). /* all cats are mammals */
    mammal(X) :- dog(X). /* all dogs are mammals */

    animal(X) :- mammal(X).
    cat(kitty).
    dog(puppy).
```

- Query:
  - cat(kitty).
  - mammal(kitty).
  - mammal(puppy).

- Show animal.pl. Ask the correctness. [wrong, should be uppercase X in the rules]

```prolog
    isCat(tom).
    isMouse(jerry).
    isMammal(X) :- isCat(X); isMouse(X).
    isAnimal(X) :- isMammal(X).
```

• Exercise:
  - Given the facts that:

```prolog
    likes(john, mary).
    likes(john, trains).
    likes(peter, fast_cars).

    hobby(john, trainspotting).
    hobby(tim, sailing).
    hobby(helen, trainspotting).
    hobby(simon, sailing).
```

- Ask students to write a rule that if person1 and person2 have the same hobby, person1 likes person2.

- Solution:

```prolog
    likes(Person1, Person2) :- hobby(Person1, Hobby), hobby(Person2, Hobby).
```

- Show exercise1.pl

- Query:

```prolog
    likes(john, trains). // true
    likes(helen, john). // true
```
likes(tim, helen). // true
likes(john, helen). // true

Recursion

• Recursion is a way to loop in Prolog.
• It allows repeatedly execute some operation till a certain point is reached or over a whole data structure.
• A recursion should have a first fact that acts as the base case.
• Then it should have some rule(s) that performs some recursive operation.

• Example (recursion1.pl)
  - To determine if there is a route to Grand Canyon.
    
    on_route(grand_canyon).
    on_route(Place) :- move(Place, _Method, NewPlace), on_route(NewPlace).
    move(home, bus, boston).
    move(boston, plane, las_vegas).
    move(las_vegas, vechicle, grand_canyon)

    - Query
      
      on_route(home).