Logic Programming Language (II)

Horn Clauses

- **Horn clauses** are named for the logician **Alfred Horn**, who first pointed out their significance in 1951.
- Horn clauses form the **basis of logic programming**.
- It has been used in logical programming, formal specification, and model theory.

- A Horn clause is a **statement** that **connects a single predicate to a set of conditions for that predicate to apply**.
- The rule, `connection(X, Y) :- friend(X, Z), friend(Z, Y)`, in the social network program is an example of the Horn clauses.
- 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.
- 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.

Resolution

- Resolution gives us the **ability** to start **searching a database** of facts by **connected together** clauses.
- As resolution proceeds, it finds values and binds 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.
  
  ```prolog
  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 ∧ B</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

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]

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

Exercise:
- Given the facts that:

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.