Introduction to Logic Programming and Prolog

What is Logic Programming?
There are many (overlapping) perspectives on logic programming:
- Computations as Deduction
- Theorem Proving
- Non-procedural Programming
- Algorithms minus Control
- A Very High Level Programming Language
- A Procedural Interpretation of Declarative Specifications

Computation as Deduction
- Logic programming offers a slightly different paradigm for computation: COMPUTATION IS LOGICAL DEDUCTION
- It uses the language of logic to express data and programs.
  
  \[
  \forall X \text{ and } Y, X \text{ is the father of } Y \text{ if } X \text{ is a parent of } Y \text{ and the gender of } X \text{ is male.}
  \]
- Current logic programming languages use first order logic (FOL) which is often referred to as first order predicate calculus (FOPC).
- The first order refers to the constraint that we can quantify (i.e. generalize) over objects, but not over functions or relations. We can express "All elephants are mammals" but not "for every continuous function f, if n < m and f(n) < 0 and f(m) > 0 then there exists an x such that n < x < m and f(x) = 0".

Theorem Proving
- Logic Programming uses the notion of an automatic theorem prover as an interpreter.
- The theorem prover derives a desired solution from an initial set of axioms.
- Note that the proof must be a "constructive" one so that more than a true/false answer can be obtained.
- E.G. The answer to \[\exists x \text{ such that } x = \sqrt{16}\] should be \[x = 4 \text{ or } x = -4\] rather than \[true\].
Non-procedural Programming

- Logic Programming languages are non-procedural programming languages.
- A non-procedural language one in which one specifies WHAT needs to be computed but not HOW it is to be done.
- That is, one specifies:
  - the set of objects involved in the computation
  - the relationships which hold between them
  - the constraints which must hold for the problem to be solved
- and leaves it up the the language interpreter or compiler to decide HOW to satisfy the constraints.

A Simple Prolog Model

- Imagine prolog as a system which has a database composed of two components:
  - FACTS - statements about true relations which hold between particular objects in the world. For example:
    - parent(adam,able): adam is a parent of able
    - parent(eve,able): eve is a parent of able
    - male(adam): adam is male.
  - RULES - statements about true relations which hold between objects in the world which contain generalizations, expressed through the use of variables. For example, the rule
    - father(X,Y) :- parent(X,Y), male(X).
    - might express:
    - for any X and any Y, X is the father of Y if X is a parent of Y and X is male.

Nomenclature and Syntax

- A prolog rule is called a clause.
- A clause has a head, a neck and a body:
  - father(X,Y) :- parent(X,Y), male(X).
  - head neck body
  - the head is a rule's conclusion.
  - The body is a rule's premise or condition.
- note:
  - read :- as IF
  - read , as AND
  - a . marks the end of input

Prolog Database

<table>
<thead>
<tr>
<th>Facts comprising the “extensional database”</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent(adam,able)</td>
</tr>
<tr>
<td>parent(adam,cain)</td>
</tr>
<tr>
<td>male(adam)</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rules comprising the “intensional database”</th>
</tr>
</thead>
<tbody>
<tr>
<td>father(X,Y) :- parent(X,Y), male(X).</td>
</tr>
<tr>
<td>sibling(X,Y) :- ...</td>
</tr>
</tbody>
</table>
**Extensional vs. Intensional**

The terms *extensional* and *intensional* are borrowed from the language philosophers use for *epistemology*.

- **Extension** refers to whatever *extends*, i.e., "is quantifiable in space as well as in time".
- **Intension** is an antonym of extension, referring to "that class of existence which may be quantifiable in time but not in space."
- **NOT intensional with a “t”, which has to do with “will, volition, desire, plan, ...”**

For KBs and DBs we use

- *extensional* to refer to that which is explicitly represented (e.g., a fact), and
- *intensional* to refer to that which is represented abstractly, e.g., by a rule of inference.

**Epistemology** is "a branch of philosophy that investigates the origin, nature, methods, and limits of knowledge."

---

### A Simple Prolog Session

```
?- assert(parent(adam,able)).
yes
?- assert(parent(eve,able)).
yes
?- assert(male(adam)).
yes
?- parent(adam,able).
yes
?- parent(adam,X).
X = able ;
yes
?- parent(X,able).
X = adam ;
no
?- parent(X,able), male(X).
X = adam ;
no
?- parent(adam,able).
yes
?- parent(adam,X).
X = able
yes
```

---

### A Prolog Session

```
?- [user].
female(eve).
parent(adam,cain).
parent(eve,cain).
father(X,Y) :- parent(X,Y), male(X).
father(X,Y) :- parent(X,Y), female(X).
parent(Who,X) :-
    father(Pa,X),
    mother(Ma,X),
    not(Who=Ma).
parent(Who,X) :-
    mother(Pa,X),
    father(Ma,X),
    not(Who=Pa).

?- [user].
    sibling(X,Y).
X = able
Y = cain ;
X = cain
Y = able ;
```

---

### Prolog Database

<table>
<thead>
<tr>
<th>Facts comprising the &quot;extensional database&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent(adam,able)</td>
</tr>
<tr>
<td>parent(adam,cain)</td>
</tr>
<tr>
<td>male(adam)</td>
</tr>
<tr>
<td>sibling(adam,able)</td>
</tr>
<tr>
<td>sibling(adam,cain)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rules comprising the &quot;intensional database&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>father(X,Y) :- parent(X,Y), male(X).</td>
</tr>
<tr>
<td>father(X,Y) :- parent(X,Y), female(X).</td>
</tr>
</tbody>
</table>
| sibling(X,Y) :- father(Pa,X),
  father(Pa,Y),
  mother(Ma,X),
  mother(Ma,Y),
  not(X=Y).                                |
How to Satisfy a Goal

Here is an informal description of how Prolog satisfies a goal (like father(adam,X)). Suppose the goal is G:

- if G = P;Q then first satisfy P, carry any variable bindings forward to Q, and then satisfy Q.
- if G = P, then if that fails, then try to satisfy Q.
- if G = not(P) then try to satisfy P. If this succeeds, then fail and if it fails, then succeed.
- if G is a simple goal, then look for a fact in the DB that unifies with G look for a rule whose conclusion unifies with G and try to satisfy its body.

Note

- two basic conditions are true, which always succeeds, and fail, which always fails.
- A comma (,) represents conjunction (i.e. and).
- A semi-colon represents disjunction (i.e. or), as in:

  grandParent(X,Y) :- grandFather(X,Y); grandMother(X,Y).

- there is no real distinction between RULES and FACTS. A FACT is just a rule whose body is the trivial condition true. That is parent(adam,cain) and parent(adam,cain) :- true. are equivalent.
- Goals can usually be posed with any of several combination of variables and constants:
  - parent(cain,able) - is Cain Able's parent?
  - parent(cain,X) - Who is a child of Cain?
  - parent(X,cain) - Who is Cain a child of?
  - parent(X,Y) - What two people have a parent/child relationship?

Terms

- The term is the basic data structure in Prolog.
- The term is to Prolog what the s-expression is to Lisp.
- A term is either:
  - a constant - e.g.
    - john, 13, 3.1415, +, 'a constant'
  - a variable - e.g.
    - X, Var, _, _foo
  - a compound term - e.g.
    - part(arm,body)
    - part(arm(john),body(john))

Compound Terms

- A compound term can be thought of as a relation between one or more terms:
  - part_of(finger,hand)
and is written as:

  - the relation name (called the principle functor) which must be a constant.
  - An open parenthesis
  - The arguments - one or more terms separated by commas.
  - A closing parenthesis.

  The number of arguments of a compound terms is called its arity.
An example

- Note the Prolog programs define relations.
- All functions are subsets of relations.
- Here’s how we might define the factorial relation in Prolog.
  
  ```prolog
  fact(1,1).
  fact(N,M) :- N1 is N-1,
  fact(N1,M1),
  M is M1*N.
  ```

Another example:

```prolog
square(X,Y) :- Y is X*X.
```