Logic Programming and Prolog

Logic programming languages are not procedural or functional:
- Specifiably relate between objects
  - `lesser(3,2)`
  - `father(trajane)`
- Separate logic from control:
  - Programs declare what facts and relations are true.
  - System determines how to use facts to solve problems.
  - System instantiates variables in order to make relation true.
- Computation: asking, testing, and evaluation (Unification, reduction, Backtracking)
- Higher-order than imperative languages

Jumping Right In

We can state these facts:

- `male(america)`
- `female(alice)`
- `male(tom)`
- `female(diana)`
- `male(trajane)`

We can then make queries:

- `male(america)`
  - Yes
- `male(trajane)`
  - Yes
- `female(diana)`
  - Yes
- `father(trajane)`
  - Yes

We can also state rules, such as this one:

- `father(X, Y) :- parent(X, Z), parent(Z, Y)`

Then the queries become more interesting:

- `father(america, tom)`
  - Yes
- `father(america, diana)`
  - Yes

Prolog vs Scheme

In Scheme, we program with functions ("procedures"):
- A function's arguments are different from the function's value.
- Give a single `Scheme` function, we can ask only one kind of question:
  - Here are the argument values; tell me what is the function's value.

In Prolog, we program with relations:
- There is no bias; all arguments are the same.
- Given a single `Prolog` predicate, we can ask many kinds of question:
  - Here are some of the argument values; tell me what the others have to be in order to make a true statement.
A Horn clause is of the form:

\[ \text{head} \leftarrow \text{body} \]

Where:
- **Head** is a single atomic formula.
- **Body** is a finite set of atomic formulae (can be empty).

**Rules**

1. **Or** - each body formula can be disjunction of literals.
2. **And** - head cannot be conjunction of literals.
3. **Negation** - no negation permitted in either part.

**Example 1:**
- Head: `m(x)`
- Body: `{ p(x), q(x) }`

**Example 2:**
- Head: `p(x)`
- Body: `{ q(x), r(x) }`

**Example 3:**
- Head: `{ s(x), t(x) }`
- Body: `{ u(x), v(x) }`

**Example 4:**
- Head: `{ a(x), b(x) }`
- Body: `{ c(x), d(x) }`

**Example 5:**
- Head: `{ e(x) }`
- Body: `{ f(x), g(x) }`

**Example 6:**
- Head: `{ h(x) }`
- Body: `{ i(x), j(x) }`
Bending the Restrictions

Getting disjoined antecedents
Example: \( a_1 \lor a_2 \lor a_3 \rightarrow c \)
Solution:

Getting more than 1 consequent, conjoined
Example: \( a_1 \land a_2 \land a_3 \rightarrow c_1 \land c_2 \)
Solution:

Getting more than 1 consequent, disjoined
Example: \( a_1 \land a_2 \land a_3 \rightarrow c_1 \lor c_2 \)
Solution:

Why Can’t We Disjoin Consequents?
Why did the designers of Prolog disallow this?

Logic Review

Horn Clauses with Variables
Variables may appear in the antecedents and consequent of a Horn clause:

- \( c(X_1, \ldots, X_n) \rightarrow h(X_1, \ldots, X_n) \)
  "For all values of \( X_1, \ldots, X_n \), the formula \( c(X_1, \ldots, X_n) \) is true if the formula \( h(X_1, \ldots, X_n) \) is true"

- \( c(X_1, \ldots, X_n) \rightarrow h(X_1, \ldots, X_n) \)
  "For all values of \( X_1, \ldots, X_n \), the formula \( c(X_1, \ldots, X_n) \) is true if there exist values of \( Y_1, \ldots, Y_m \) such that the formula \( h(X_1, \ldots, X_n, Y_1, \ldots, Y_m) \) is true"

Meaning of Prolog Rules With Variables

Example:

\( \text{isMother}(X) :\equiv \text{female}(X), \text{parent}(X, Y) \)

Logic:

\( \text{parent}(X, Y) \land \text{female}(X) \equiv \text{isMother}(X), \)

But this is meaningless without quantifiers for the variables.

The rule

A Prolog rule of this form \( \exists X_1, \ldots, X_n \exists Y_1, \ldots, Y_m : a(X_1, \ldots, X_n) \rightarrow b(Y_1, \ldots, Y_m) \)

Sample run

```

cd /pl
val.pl to SWProlog (release-branching, version 5.2.1),
 filetype: source, variables: [Y]
 설정된 변수: [Y]
家族 크기 0.00 sec, 0.008 bytes
氏族の人数: 0.00 sec, 0.008 bytes
Y = isMother(X),
X = victoria,
X = victoria,
Y =
```

Rule Ordering and Unification

1. rule ordering used in search
2. unification requires two instances of the same variable in the same rule to get the same value
3. unification does not require differently named variables to get different values: hence, sibling(Edward, Edward),
4. all rules searched if requested by ;

How Prolog Handles a Query

Example 1

Database:

1) male(tina),
2) male(mike),
3) male(doug),
4) female(victoria),
5) male(david),
6) parent(doug, ryan),
7) parent(tom, william),
8) parent(doug, eddie),
9) parent(doug, ted),
10) parent(tina, mark),

Query:

1) \( \text{grandfather}(X, Y) \),

Trace it by hand

Trace it in Prolog

The Anonymous Variable

Example 2

Procedural Semantics of Prolog
Logic Programming vs. Prolog

cousin(X,Y) :- parent(W,X), sister(W,Z),
               parent(Z,Y).

cousin(X,Y) :- parent(W,X), brother(W,Z),
               parent(Z,Y).

|?- cousin(X,jane), % a query

Rule and Goal Ordering:
- There are two rules for cousin
- Which rule do we try first?
- Each rule for cousin has several subgoals
- Which subgoal do we try first?

Logic Programming: 
- Non-deterministic
  - Arbitrarily choose rule to expand first
  - Arbitrarily choose subgoal to explore first
  - Results don't depend on rule and subgoal ordering

Prolog: Deterministic
- Expand first rule first
- Explore first subgoal first
- Results may depend on rule and subgoal ordering