Chapter 12 - Logic Programming

Programming Languages:

Principles and Practice, 2nd Ed.

Kenneth C. Louden

Logic Programming

- **Uses a set of logical assertions (i.e. statements that are either true or false), as a program (the facts).**
- **Execution is initiated by a *query* or *goal*, which the system attempts to prove true or false, based on the existing set of assertions.**
- **For this reason, logic programming systems are sometimes called *deductive databases*.**
- **Two main "weirdnesses": no explicit functions, no explicit execution control.**

Examples

- **Computing ancestors:**
  A parent is an ancestor.
  If A is an ancestor of B, and B is an ancestor of C, then A is an ancestor of C. (a typical relation: called ??)
  A mother is a parent.
  A father is a parent.
  Bill is the father of Jill.
  Jill is the mother of Sam.
  Bob is the father of Sam.

Note incompleteness!

- **Computing the factorial function:**
  The factorial of 0 is 1.
  If m is the factorial of n - 1, then n * m is the factorial of n.

(First order) Predicate Calculus

Starts with a set of axioms (true assertions), stated using the following elements:

- ***Constants.*** **Usually numbers or names. In the examples, Bill and 0 are constants.**
- ***Predicates.*** **Names for functions that are true or false, like Boolean functions in a program. Predicates can take a number of arguments. In the examples, ancestor and factorial are predicates.**
- ***Functions.*** **First-order predicate calculus distinguishes between functions that are true or false—these are the predicates—and all other functions, which represent non-Boolean values. * is a function in the examples. (Factorial - a function - is indirectly expressed as a predicate.)**

Predicate Calculus (continued)

- ***Variables*** **that stand for as yet unspecified quantities. In the examples, A and m are variables.**
- ***Connectives.*** **Operations and, or, and not; implication "→" and equivalence " ↔" (derivable from the previous three).**
- ***Quantifiers.*** **These are operations that introduce variables: "for all" - the *universal quantifier*, and "there exists" - the *existential quantifier.***
- ***Punctuation symbols:*** **left and right parentheses, the comma, and the period.**

Note: Arguments to predicates and functions can only be terms: combinations of variables, constants, and functions.

Examples written in Pred. Calc.:

- **Ancestors:**
  For all X and Y, parent(X,Y) $\rightarrow$ ancestor(X,Y).
  For all A, B, and C, ancestor(A,B) and ancestor(B,C) $\rightarrow$ ancestor(A,C).
  For all X and Y, mother(X,Y) $\rightarrow$ parent(X,Y).
  For all X and Y, father(X,Y) $\rightarrow$ parent(X,Y).
  Father(Bill,Jill).
  Mother(Jill,Sam).
  Father(Bob,Sam).
- **Factorials:**
  Factorial(0,1).
  For all n and m, factorial(n-1,m) $\rightarrow$ factorial(n,n*m).

Horn Clauses

Drop the quantifiers (i.e., assume them implicitly). Distinguish variables from constants, predicates, and functions by upper/lower case:

- **parent(X,Y) $\rightarrow$ ancestor(X,Y).**
  ancestor(A,B) and ancestor(B,C) $\rightarrow$ ancestor(A,C).
  mother(X,Y) $\rightarrow$ parent(X,Y).
  father(X,Y) $\rightarrow$ parent(X,Y).
  father(bill,jill).
  mother(jill,sam).
  father(bob,sam).
- **factorial(0,1).**
  factorial(N-1,M) $\rightarrow$ factorial(N,N*M).

Prolog

Modified Horn clause syntax: write the clauses backward, with :- as the (backward) arrow, comma as "and" and semicolon as "or":

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :-
    ancestor(X,Z), ancestor(Z,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
factorial(0,1).
factorial(N,N*M) :- factorial(N-1,M).
```

Unfortunate Errors:

```
?- ancestor(bill,sam).
Yes.
?- ancestor(bill,X).
X = jill ;
X = sam ;
ERROR: Out of local stack
?- ancestor(X,bob).
ERROR: Out of local stack
?- factorial(2,2).
No
?- factorial(0,X).
X = 1 ;
ERROR: Out of global stack
```

What's Wrong????

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :-
    ancestor(X,Z), ancestor(Z,Y).
parent(X,Y) :- mother(X,Y).
parent(X,Y) :- father(X,Y).
father(bill,jill).
mother(jill,sam).
father(bob,sam).
factorial(0,1).
factorial(N,N*M) :- factorial(N-1,M).
```

Immediate recursion

No arithmetic is actually computed

Arithmetic

- **Easy to fix - arithmetic expressions must be explicitly forced:**

```
?- 2*3 = 6.
No
?- 2*3 = 2*3.
Yes
?- 6 is 2*3.
Yes
?- 2*3 is 6.
No
```

"is" forces its right-hand argument

But not its left-hand argument

- **Rewrite factorial like this:**

```
factorial(0,1).
factorial(N,P) :- N1 is N-1,
    factorial(N1,M), P is M*N.
```

Stack Problems a bit harder:

- **Stack problems still exist for factorial:**

```
?- factorial(7,X).
X = 5040 ;
ERROR: Out of local stack
```

- **To figure this out, we must understand the procedural interpretation of Horn clauses:**

```
a :- b, c, d.
```

  represents the definition of a (boolean) function named a. The body of a is given by the clauses on the right hand side: b, then c, then d.

- **Note similarity to recursive-descent parsing! Also note sequencing!**

Prolog's Execution Strategy

- **Given a query or goal, Prolog tries to pattern match the goal with the left-hand sides of all clauses, in a sequential top-down fashion.**
- **Any lhs that matches causes the rhs terms to be set up sequentially as *subgoals*, which Prolog immediately tries to match in turn with lhs terms.**
- **Thus, Prolog's execution path is top-down, left-to-right, depth-first. All intermediate results are kept for backtracking purposes.**

Solving the stack problems

- **For the ancestor problem, just remove the left recursion:**

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :-
    ~~ancestor~~(X,Z), ancestor(Z,Y).
    parent(X,Z)
```

- **For the factorial problem, it involves *preventing* reuse of the recursive rule:**

```
factorial(0,1).
factorial(N,P) :- not(N=0), N1 is N-1,
     factorial(N1,M), P is M*N.
```

Lists

- **Prolog uses almost the same list syntax as Haskell or ML: [1,2,3]**
- **Head and tail pattern syntax is different: [H|T] versus (h:t) in Haskell or (h::t) in ML**
- **Also, as many elements as desired can be written using commas before the bar: [1,2,3|[4]] is the list [1,2,3,4].**
- **Example:**

```
?- [1,2,3] = [X,Y|_].
X = 1
Y = 2
```

Pattern Matching

- **Pattern matching is more general in Prolog than it is in Haskell:**
  - **variables can be repeated in patterns, implying that they must be the same value:**
```
?- [1,1] = [X,X].
X = 1
?- [1,2] = [X,X].
No
```
  - **Other operations can be more general too.**

Final Examples

- **The append list operation:**
```
append([],X,X).
append([H|T],L,[H|M]) :-
                   append(T,L,M).
```
- **Quicksort:**
```
quicksort([],[]).
quicksort([H|T],Result) :-
  split(H,T,A,B),/* exercise */
  quicksort(A,X),
  quicksort(B,Y),
  append(X,[H|Y],Result).
```