Programming Languages

2nd edition
Tucker and Noonan

Chapter 15
Logic Programming

Q: How many legs does a dog have if you call its tail a leg?
A: Four. Calling a tail a leg doesn't make it one.
Abraham Lincoln

Contents

15.2.2 Practical Aspects of Prolog

- Tracing
- The Cut
- Negation
- The is, not, and Other Operators
- The Assert Function

Tracing

To see the dynamics of a function call, the trace function can be used. E.g., if we want to trace a call to the following function:

```
factorial(0, 1).
factorial(N, Result) :- N > 0, M is N - 1,
                        factorial(M, SubRes), Result is N * SubRes.
```

we can activate trace and then call the function:

```
?- trace(factorial/2).
?- factorial(4, X).
```

Note: the argument to trace must include the function's arity.

Tracing Output

```
?- factorial(4, X).
Call: ( 7) factorial(4, _G173)
Call: ( 8) factorial(3, _L131)
Call: ( 9) factorial(2, _L144)
Call: ( 10) factorial(1, _L157)
Call: ( 11) factorial(0, _L170)
Exit: ( 11) factorial(0, 1)
Exit: ( 10) factorial(1, 1)
Exit: ( 9) factorial(2, 2)
Exit: ( 8) factorial(3, 6)
Exit: ( 7) factorial(4, 24)

X = 24
```

These are temporary variables

These are levels in the search tree

The Cut

The *cut* is an operator (!) inserted on the right-hand side of a rule.

semantics: the cut forces those subgoals not to be retried if the right-hand side succeeds once.

E.g (bubble sort):

```
bsort(L, S) :- append(U, [A, B | V], L),
               B < A, !,
               append(U, [B, A | V], M),
               bsort(M, S).
bsort(L, L).
```

So this code gives one answer rather than many.

Bubble Sort Trace

```
?- bsort([5,2,3,1], Ans).
Call:  ( 7) bsort([5, 2, 3, 1], _G221)
Call:  ( 8) bsort([2, 5, 3, 1], _G221)
…
Call:  ( 12) bsort([1, 2, 3, 5], _G221)
Redo:  ( 12) bsort([1, 2, 3, 5], _G221)
…
Exit:  ( 7) bsort([5, 2, 3, 1], [1, 2, 3, 5])

Ans = [1, 2, 3, 5] ;

No
```

Without the cut, this would have given some wrong answers.

The *is* Operator

is instantiates a temporary variable. E.g., in

```
factorial(0, 1).
factorial(N, Result) :- N > 0, M is N - 1,
                        factorial(M, SubRes), Result is N * SubRes.
```

Here, the variables M and Result are instantiated This is like an assignment to a local variable in C-like languages.

Other Operators

Prolog provides the operators

+ - * / ^ = < > >= =< \=

with their usual interpretations.

The not operator is implemented as goal failure. E.g.,

```
factorial(N, 1) :- N < 1.
factorial(N, Result) :- not(N < 1), M is N - 1,
                        factorial(M, P),
                        Result is N * P.
```

is equivalent to using the cut (!) in the first rule.

The assert Function

The assert function can update the facts and rules of a program dynamically. E.g., if we add the following to the foregoing database program:

```
?- assert(mother(jane, joe)).
```

Then the query:

```
?- mother(jane, X).
```

gives:

```
X = ron ;
X = joe;
No
```


15.3 Prolog Examples

1. Symbolic Differentiation
   Symbol manipulation and logical deduction united
2. Solving Word Problems
   Nondeterminism seeks all solutions, not just one
3. Natural Language Processing
   One of Prolog's traditional research applications
4. Semantics of Clite
   Declarative languages help model rapid designs
5. Eight Queens Problem
   Exploiting Prolog's natural backtracking mechanism

15.3.1 Symbolic Differentiation

Symbolic Differentiation Rules
Fig 15.9

$$\frac{d}{dx}(c) = 0 \qquad c \text{ is a constant}$$

$$\frac{d}{dx}(x) = 1$$

$$\frac{d}{dx}(u+v) = \frac{du}{dx} + \frac{dv}{dx} \qquad u \text{ and } v \text{ are functions of } x$$

$$\frac{d}{dx}(u-v) = \frac{du}{dx} - \frac{dv}{dx}$$

$$\frac{d}{dx}(uv) = u\frac{dv}{dx} + v\frac{du}{dx}$$

$$\frac{d}{dx}(u/v) = \left(v\frac{du}{dx} - u\frac{dv}{dx}\right)/v^2$$

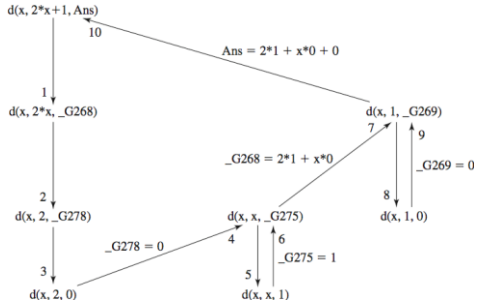
Prolog Encoding

1. Uses Infix notation
   E.g., 2x + 1 is written as 2*x+1
2. Function d incorporates these rules.
   E.g., d(x,2*x+1, Ans) should give an answer.
3. However, no simplification is performed.
   E.g. the answer for d(x,2*x+1, Ans) is
   2*1+x*0+0
   which is equivalent to the simplified answer, 2.

Prolog Program

```
d(X, U+V, DU+DV) :- d(X, U, DU), d(X, V, DV).
d(X, U-V, DU-DV) :- d(X, U, DU), d(X, V, DV).
d(X, U*V, U*DV + V*DU) :- d(X, U, DU), d(X, V, DV).
d(X, U/V, (V*DU - U*DV)/(V*V)) :- d(X, U, DU), d(X, V, DV).
d(X, C, 0) :- atomic(C), C\=X.
d(X, X, 1).
```


Search Tree for d(x, 2*x+1, Ans)



15.3.2 Solving Word Problems

A simple example:

> *Baker, Cooper, Fletcher, Miller, and Smith live in a five-story building. Baker doesn't live on the 5th floor and Cooper doesn't live on the first. Fletcher doesn't live on the top or the bottom floor, and he is not on a floor adjacent to Smith or Cooper. Miller lives on some floor above Cooper. Who lives on what floors?*

We can set up the solution as a list of five entries:
[floor(_,5), floor(_,4), floor(_,3), floor(_,2), floor(_,1)]

The *don't care* entries are placeholders for the five names.

Modeling the solution

We can identify the variables B, C, F, M, and S with the five persons,
and the structure floors(Floors) as a function whose argument is the list to be solved.

Here's the first constraint:
member(floor(baker, B), Floors), B\=5
which says that *Baker doesn't live on the 5th floor*.

The other four constraints are coded similarly, leading to the following program:

Prolog solution

```
floors([floor(_,5),floor(_,4),floor(_,3),floor(_,2),floor(_,1)]).
building(Floors) :- floors(Floors),
   member(floor(baker, B), Floors), B \= 5,
   member(floor(cooper, C), Floors), C \= 1,
   member(floor(fletcher, F), Floors), F \= 1, F \= 5,
   member(floor(miller, M), Floors), M > C,
   member(floor(smith, S), Floors), not(adjacent(S, F)),
   not(adjacent(F, C)),
   print_floors(Floors).
```

Auxiliary functions

Floor adjacency:

```
adjacent(X, Y) :- X =:= Y+1.
adjacent(X, Y) :- X =:= Y-1.
```

Note: =:= tests for numerical equality.

Displaying the results:

```
print_floors([A | B]) :- write(A), nl, print_floors(B).
print_floors([]).
```

Note: write is a Prolog function and nl stands for "new line."

Solving the puzzle is done with the query:

```
?- building(X).
```

which finds an instantiation for X that satisfies all the constraints.