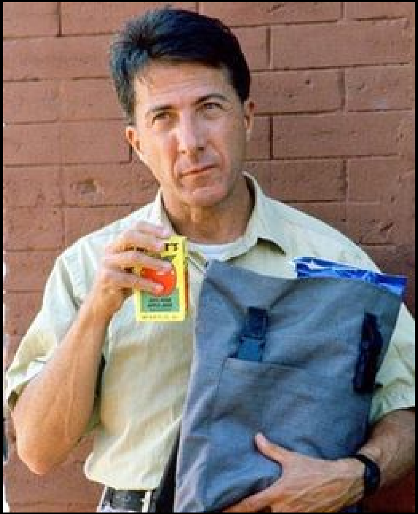
Seven Languages in Seven Weeks

Correl Roush

June 24, 2015



Created	1972
Author	Alain Colmerauer and Phillipe Roussel

A declarative logic programming language.

GNU Prolog

`http://www.gprolog.org/`

SWI Prolog

`http://www.swi-prolog.org/`

- Atoms & Variables
- Facts & Rules
- Unification

Atoms

- Begin with a lowercase letter.

Variables

- Begin with an uppercase letter.

Facts

```
likes(wallace, cheese).
likes(grommit, cheese).
likes(wendolene, sheep).
```

Rules

```
friend(X, Y) :- \+(X = Y),
                likes(X, Z),
                likes(Y, Z).
```

Example (Queries)

```
likes(wallace, sheep).
%% false

likes(grommit, cheese).
%% true

friend(grommit, wallace).
%% true

friend(wallace, grommit).
%% true

friend(wendolene, grommit).
%% false
```

Facts

```
food_type(velveeta, cheese).
food_type(ritz, cracker).
food_type(spam, meat).
food_type(sausage, meat).
food_type(jolt, soda).
food_type(twinkie, dessert).

flavor(sweet, desert).
flavor(savory, meat).
flavor(savory, cheese).
flavor(sweet, soda).
```

Rules

```
food_flavor(X, Y) :- food_type(X, Z),
                     flavor(Y, Z).
```

Example (Queries)

```
food_type(What, meat).
%% What = spam ;
%% What = sausage.

food_flavor(sausage, sweet).
%% false.

flavor(sweet, What).
%% What = dessert ;
%% What = soda.

food_flavor(What, savory).
%% What = velveeta ;
%% What = spam ;
%% What = sausage.
```

MAP COLORING: PROBLEM

- We want to color a map of the southeastern United States.
- We do not want two states of the same color to touch.
- We will use three colors: red, blue, and green.

Facts

```
different(red, green). different(red, blue).
different(green, red). different(green, blue).
different(blue, red). different(blue, green).

coloring(Alabama, Mississippi,
         Georgia, Tennessee, Florida) :-
  different(Mississippi, Tennessee),
  different(Mississippi, Alabama),
  different(Alabama, Tennessee),
  different(Alabama, Mississippi),
  different(Alabama, Georgia),
  different(Alabama, Florida),
  different(Georgia, Florida),
  different(Georgia, Tennessee).
```

Example (Query)

```
coloring(Alabama, Mississippi,
         Georgia, Tennessee, Florida).
%% Alabama = blue,
%% Florida = green,
%% Georgia = red ,
%% Mississippi = red,
%% Tennessee = green ;
```

Unification

Unification across two structures tries to make both structures identical.

Facts

```
cat(lion).
cat(tiger).
```

Rules

```
dorothy(X, Y, Z) :- X = lion,
                    Y = tiger,
                    Z = bear.
twin_cats(X, Y) :- cat(X), cat(Y).
```

Example (Unification)

```
dorothy(lion, tiger, bear).
%% true.

dorothy(One, Two, Three).
%% One = lion,
%% Two = tiger,
%% Three = bear.

twin_cats(One, Two).
%% One = lion,
%% Two = lion ;
%% One = lion,
%% Two = tiger ;
%% One = tiger,
%% Two = lion ;
%% One = tiger,
%% Two = tiger.
```

An interview with Brian Tarbox, Dolphin Researcher

EXERCISES

- Recursion
- Lists and Tuples
- Unification
- Lists and Math
- Using rules in Both Directions

The following rules define the paternal family tree of the Waltons. They express a father relationship and from that infers the ancestor relationship. Since an ancestor can mean a father, grandfather, or great grandfather, we will need to nest the rules or iterate.

```
father(zeb,         john_boy_sr).
father(john_boy_sr, john_boy_jr).

ancestor(X, Y) :-
    father(X, Y).
ancestor(X, Y) :-
    father(X, Z), ancestor(Z, Y).
```

In the above example, `ancestor(Z, Y)` is a recursive subgoal.

- Lists are containers of variable length.
- Tuples are containers with a fixed length.

Tuples unify if they have the same number of elements, and each element unifies.

```
(1, 2, 3) = (1, 2, 3).    %% true
(1, 2, 3) = (1, 2, 3, 4). %% false
(1, 2, 3) = (3, 2, 1).    %% false
```

Lists behave similarly, but can be deconstructed with the pattern `[Head|Tail]`.

```
[1, 2, 3] = [1, 2, 3].             %% true
[2, 2, 3] = [X, X, Z].             %% X = 2, Z = 3

[a, b, c] = [Head|Tail].           %% Head = a, Tail = [b, c]
[] = [Head|Tail].                  %% false
[a] = [Head|Tail].                 %% Head = a, Tail = []

[a, b, c] = [a|[Head|Tail]].       %% Head = b, Tail = [c]

[a, b, c, d, e] = [_, _|[Head|_]]. %% Head = c
```

Count

```
count(0, []).
count(Count, [Head|Tail]) :- count(TailCount, Tail), Count is TailCount + 1.
```

Sum

```
sum(0, []).
sum(Total, [Head|Tail]) :- sum(Sum, Tail), Total is Head + Sum.
```

Average

```
average(Average, List) :- sum(Sum, List), count(Count, List), Average is Sum/Count.
```

The rule `append(List1, List2, List3)` is true if `List3` is `List1` + `List2`.

… as a lie detector

```
append([oil], [water],
       [oil, water]). %% true
append([oil], [water],
       [oil, slick]). %% false
```

… as a list builder

```
append([tiny], [bubbles],
       What).
%% What = [tiny, bubbles]
```

… for list subtraction

```
append([dessert_topping], Who,
       [dessert_topping, floor_wax]).
%% Who = [floor_wax]
```

… for computing possible splits

```
append(One, Two,
       [apples, oranges, bananas]).

%% One = [], Two = [apples, oranges, bananas]
%% One = [apples], Two = [oranges, bananas]
%% One = [apples, oranges], Two = [bananas]
%% One = [apples, oranges, bananas], Two = []
```

Steps:

- Write a rule called `concatenate(List1, List2, List3)` that can concatenate an empty list to `List1`.
- Add a rule that concatenates one item from `List1` onto `List2`.
- Add a rule that concatenates two and three items from `List1` onto `List2`.
- See what we can generalize.

`concatentate/3` is `true` if the first parameter is an empty list and the next two parameters are the same.

```
concatenate([], List, List).
```

Example (Test)

```
concatenate([], [harry], What).
%% What = [harry]
```

Add a rule that concatenates the first element of `List1` tot he front of `List2`:

```
concatenate([Head|[]], List, [Head|List]).
```

Example (Test)

```
concatenate([malfoy], [potter], What).
%% What = [malfoy, potter]
```

Define another couple of rules to concatenate lists of lengths 2 and 3:

```
concatenate([Head1|[Head2|[]]], List, [Head1, Head2|List]).
concatenate([Head1|[Head2|[Head3|[]]]], List, [Head1, Head2, Head3|List])
```

Example (Test)

```
concatenate([malfoy, granger], [potter], What).
%% What = [malfoy, granger, potter]
```

Generalize for lists of arbitrary length using nested rules:

```
concatenate([], List, List).
concatenate([Head|Tail1], List, [Head|Tail2]) :-
    concatenate(Tail1, List, Tail2).
```

EXERCISES

- Sudoku
- Eight Queens

- For a solved puzzle, the numbers in the puzzle and solution should be the same.
- A Sudoku board is a grid of sixteen cells, with values from 1-4.
- The board has four rows, four columns, and four squares.
- A puzzle is valid if the elements in each row, column, and square has no repeated elements.

Example (Example)

```
sudoku([_, _, 2, 3,
        _, _, _, _,
        _, _, _, _,
        3, 4, _, _],
       Solution).
```

SOLVING SUDOKU: THE SOLUTION

```
valid([]).
valid([Head|Tail]) :-
    fd_all_different(Head),
    valid(Tail).

sudoku(Puzzle, Solution) :-
        Solution = Puzzle,
        Puzzle = [S11, S12, S13, S14,
                  S21, S22, S23, S24,
                  S31, S32, S33, S34,
                  S41, S42, S43, S44],

        fd_domain(Solution, 1, 4),

        Row1 = [S11, S12, S13, S14],
        Row2 = [S21, S22, S23, S24],
        Row3 = [S31, S32, S33, S34],
        Row4 = [S41, S42, S43, S44],

        Col1 = [S11, S21, S31, S41],
        Col2 = [S12, S22, S32, S42],
        Col3 = [S13, S23, S33, S43],
        Col4 = [S14, S24, S34, S44],

        Square1 = [S11, S12, S21, S22],
        Square2 = [S13, S14, S23, S24],
        Square3 = [S31, S32, S41, S42],
        Square4 = [S33, S34, S43, S44],

        valid([Row1, Row2, Row3, Row4,
               Col1, Col2, Col3, Col4,
               Square1, Square2, Square3, Square4]).
```

- A board has eight queens.
- Each queen has a row from 1-8 and a column from 1-8.
- No two queens can share the same row.
- No two queens can share the same column.
- No two queens can share the same diagonal (southwest to northeast).
- No two queens can share the same diagonal (northwest to southeast).

EIGHT QUEENS: THE SOLUTION

```
valid_queen((Row, Col)) :-
    member(Col, [1,2,3,4,5,6,7,8]).
valid_board([]).
valid_board([Head|Tail]) :-
    valid_queen(Head), valid_board(Tail).

cols([], []).
cols([(_, Col)|QueensTail], [Col|ColsTail]) :-
    cols(QueensTail, ColsTail).

diags1([], []).
diags1([(Row, Col)|QueensTail],
       [Diagonal|DiagonalsTail]) :-
    Diagonal is Col - Row,
    diags1(QueensTail, DiagonalsTail).

diags2([], []).
diags2([(Row, Col)|QueensTail],
       [Diagonal|DiagonalsTail]) :-
    Diagonal is Col + Row,
    diags2(QueensTail, DiagonalsTail).

eight_queens(Board) :-
    Board = [(1, _), (2, _), (3, _), (4, _),
             (5, _), (6, _), (7, _), (8, _)],
    valid_board(Board),

    cols(Board, Cols),
    diags1(Board, Diags1),
    diags2(Board, Diags2),
    fd_all_different(Cols),
    fd_all_different(Diags1),
    fd_all_different(Diags2).
```

EXERCISES

WRAPPING UP PROLOG: STRENGTHS

- Natural-Language Processing
- Games
- Semantic Web
- Artificial Intelligence
- Scheduling

- Utility
- Very Large Data Sets
- Mixing the Imperative and Declarative Models

Prolog was a particularly poignant example of my evolving understanding. If you find a problem that's especially well suited for Prolog, take advantage. In such a setting, you can best use this rules-based language in combination with other general-purpose languages, just as you would use SQL within Ruby or Java.