# T-93.540 Logic Programming
# Logic Programming Assignment
# Rubik's Cube

Simo Neuvonen
48002K, sneuvone@cc.hut.fi

21st December 2001

## 1 Introduction

The subject of the assignment is to create a logic program (with Prolog) that solves the Rubik's cube. The 3x3x3 cube was choosed because of the own interest of the author, although the solution for the 2x2x2 cube would probably have been easier to implement.

### 1.1 Assignment Instructions

"The subject of the assignment is to create a program that solves the Rubik's Cube. If the subject seems to be too time-consuming, create a program that solves the 2x2x2 minicube."

## 2 Description of the problem

### 2.1 Solving the Rubik's Cube with Prolog

The main idea for the logic program solving the Rubik's Cube is to use a heuristic search that tries a huge amount of move combinations, trying to find the solution. The cube must be solved one piece at time, so that the full solution consists of multiple partial-solutions. Prolog features, like unification and backtracking are used when matching the cube state against the partial-goal and when trying new moves to improve the cube.

### 2.2 Algorithms used

The most important algorithm in the program is a Breadth-First-Search (BFS), which tries to find a solution for a given partial-problem. The BFS is feeded with improved partial-goals, which are actually the desired target states of the cube after each sequence of the rotations. When a partial solution is found, the moves are saved and applied, and the program moves to the next stage.

This program relies on the solving method presented in [2]. That is, the cube is solved in stages with a certain partial goals. These are:

```
stages   1-4: put the upper slice edge pieces (two-colored) in place
stages   5-8: put the upper slice corner pieces (three-colored) in place
stage      9: turn the cube so that the 'F' face center and the upper
              slice 'F' face pieces are on the same (front) face
stages 10-20: put the middle slice corner pieces (two-colored) in place
stages 21-24: form a one-color "cross" on the 'D' face with bottom slice
              (two-colored) edge pieces
stages 25-27: put the bottom slice edge pieces (two-colored) in place
stage     28: put the bottom slice corner pieces in place
```

Ideas and program stubs for the implementation of the predicates and data structures were picked from [1].

## 3 Predicates and data structures

### 3.1 Predicates

| | |
|---|---|
| `mov` | performs the actual changes in the cube, moves list items from one place to another |
| `move` | interface, that uses mov either in clockwise or counterclockwise direction |
| `move_sequence` | reads a list of moves and applies them to the cube |
| `get_goal` | gives cube goal state to search a sequence of moves for |
| `cand` | lists the allowed moves |
| `get_candidate` | fetches a move from the candidate move list |
| `conflict` | checks that the move under investigation is not dummy |
| `rotate` | performs a Breadth-First-Search with the given cube state and a goal state to find the needed move_sequence |
| `get_stage` | updates the stage number |
| `stage` | calls rotate for each stage, saves the partial solution and then continues the search |
| `solve` | calls stage with stage parameter predefined to starting stage |

### 3.2 Data structures

A cube has 6*9=54 tiles (tile = one color face of the 1x1x1 piece), which are presented as a list in the program. The tiles are named according to which face of the 3x3x3 cube they belong in the solved cube (you can think colors instead of direction, if you prefer). So, the names of faces of the cube are:

```
F - front face
R - right face
B - back face
L - left face
U - up face
D - down face
```

Each of the cube face has 9 tiles, each of which has the same face name. The tile list can be collected by reading the cube faces in the same order as above (F-R-B-L-U-D). The upper left tile of each face comes always first, then other tiles of the same face line-by-line from left to right.

## 3.3 Notation

The notation used in the program is quite straightforward: moves are named with letters:

```
u - rotate the upper slice
d - rotate the bottom slice
l - rotate the left slice
r - rotate the right slice
f - rotate the front slice
b - rotate the back slice
t - turn the cube around vertical axis
```

A prefix tells which direction to rotate, + means forward (l,r) or clockwise (f, b, u, d, t), - means backwards (l, r) or counterclockwise (f, b, u, d, t).

The moves are returned as a list. For example, a list [+r, -f, +t] means that the first the right slice must be rotated forward, then the front slice to the left and finally the whole cube must be rotated 90 degrees clockwise.

Names of the special move sequences consist of the prefix 'sp' and the number. Here is the complete list of the special moves:

```
sp1      [-f, +d, +f, -d, -l, -d, +l]
sp2      [+f, -d, -f, +d, -r, +d, +r]
sp3      [-b, +r, -d, -r, +d, +b]
sp4      [-b, -d, +r, +d, -r, +b]
sp5      [-b, +r, -d, -r, +d, +b, +f, -d, -l, +d, +l, -f]
sp6      [-l, -l, -u, -d, -d, +l, +r, +f, +f, -l, -r, +u, -l, -l]
sp7      [-l, -l, -u, +l, +r, +f, +f, -l, -r, -d, -d, +u, -l, -l]

sp8      [-l, -l, +u, +u, -b, -b, +u, +r, +r, +f, +f, +r, +r, +f,
          +f, +r, +r, +f, +f, -u, -b, -b, +u, +u, -l, -l]

sp9      [-r, -d, -l, +d, +r, -d, +l, +d]
sp10     [-d, -l, +d, -r, -d, +l, +d, +r]
sp11     [+l, -b, -b, +f, -l, -f, +l, +f, -l, -f, +l, +f, -l, -f, +l, -b, -b, -l]
sp12     [-b, +r, -d, -r, +d, +r, -d, -r, +d, +r, -d, -r, +d, +b]
sp13     [+f, +u, -f, -u, +f, +u, -f, -u]
sp14     [+u, +f, -u, -f, +u, +f, -u, -f]
```

# 4 The solution

## 4.1 Features

## 4.2 Performance

The example cube was solved by the program in about three minutes, which is actually quite a bearable time. The test was run on a Niksula's Sun UltraSparc 10. However, in complex situations, the search may last much longer, when the move sequences grow large.

When tracing the search process, the most time-consuming stages were the early stages 6-8, as expected. Putting the upmost level slice pieces into correct places sometime demands long move sequences.

## 5 Code listing

```
% Rubik's Cube Solver
% by Simo Neuvonen, sneuvone@cs.hut.fi
%
% last modified 2001-12-21
%
% References: Merrit,Dennis: "Prolog In Action: Solving Rubik's Cube",
%             <http://www.amzi.com/articles/rubik.htm>
%
%             Jeays, Mark: "How to Solve Rubik's Cube 2.3",
%             <http://jeays.net/rubiks.htm>
%
% ---------------------------------------------------------------
% the actual rotations are performed here
% ---------------------------------------------------------------

% rotate "right" slice

mov(r,
cube(
X1,X2,X3,X4,X5,X6,X7,X8,X9,
X10,X11,X12,X13,X14,X15,X16,X17,X18,
X19,X20,X21,X22,X23,X24,X25,X26,X27,
X28,X29,X30,X31,X32,X33,X34,X35,X36,
X37,X38,X39,X40,X41,X42,X43,X44,X45,
X46,X47,X48,X49,X50,X51,X52,X53,X54),

cube(
X1,X2,X48,X4,X5,X51,X7,X8,X54,
X16,X13,X10,X17,X14,X11,X18,X15,X12,
X45,X20,X21,X42,X23,X24,X39,X26,X27,
X28,X29,X30,X31,X32,X33,X34,X35,X36,
X37,X38,X3,X40,X41,X6,X43,X44,X9,
X46,X47,X25,X49,X50,X22,X52,X53,X19)
    ).

% rotate "left" slice

mov(l,
cube(
X1,X2,X3,X4,X5,X6,X7,X8,X9,
X10,X11,X12,X13,X14,X15,X16,X17,X18,
X19,X20,X21,X22,X23,X24,X25,X26,X27,
X28,X29,X30,X31,X32,X33,X34,X35,X36,
X37,X38,X39,X40,X41,X42,X43,X44,X45,
X46,X47,X48,X49,X50,X51,X52,X53,X54),

cube(
X46,X2,X3,X49,X5,X6,X52,X8,X9,
X10,X11,X12,X13,X14,X15,X16,X17,X18,
X19,X20,X37,X22,X23,X40,X25,X26,X43,
X30,X33,X36,X29,X32,X35,X28,X31,X34,
X1,X38,X39,X4,X41,X42,X7,X44,X45,
X27,X47,X48,X24,X50,X51,X21,X53,X54)
    ).

% rotate "up" slice

mov(u,
cube(
X1,X2,X3,X4,X5,X6,X7,X8,X9,
X10,X11,X12,X13,X14,X15,X16,X17,X18,
X19,X20,X21,X22,X23,X24,X25,X26,X27,
X28,X29,X30,X31,X32,X33,X34,X35,X36,
X37,X38,X39,X40,X41,X42,X43,X44,X45,
X46,X47,X48,X49,X50,X51,X52,X53,X54),

cube(
X10,X11,X12,X4,X5,X6,X7,X8,X9,
X19,X20,X21,X13,X14,X15,X16,X17,X18,
X28,X29,X30,X22,X23,X24,X25,X26,X27,
X1,X2,X3,X31,X32,X33,X34,X35,X36,
X43,X40,X37,X44,X41,X38,X45,X42,X39,
X46,X47,X48,X49,X50,X51,X52,X53,X54)
    ).

% rotate "down" slice

mov(d,
cube(
X1,X2,X3,X4,X5,X6,X7,X8,X9,
X10,X11,X12,X13,X14,X15,X16,X17,X18,
X19,X20,X21,X22,X23,X24,X25,X26,X27,
X28,X29,X30,X31,X32,X33,X34,X35,X36,
X37,X38,X39,X40,X41,X42,X43,X44,X45,
X46,X47,X48,X49,X50,X51,X52,X53,X54),

cube(
X1,X2,X3,X4,X5,X6,X16,X17,X18,
X10,X11,X12,X13,X14,X15,X25,X26,X27,
X19,X20,X21,X22,X23,X24,X34,X35,X36,
X28,X29,X30,X31,X32,X33,X7,X8,X9,
X37,X38,X39,X40,X41,X42,X43,X44,X45,
X48,X51,X54,X47,X50,X53,X46,X49,X52)).

% rotate "front" slice

mov(f,
cube(
X1,X2,X3,X4,X5,X6,X7,X8,X9,
X10,X11,X12,X13,X14,X15,X16,X17,X18,
X19,X20,X21,X22,X23,X24,X25,X26,X27,
X28,X29,X30,X31,X32,X33,X34,X35,X36,
X37,X38,X39,X40,X41,X42,X43,X44,X45,
X46,X47,X48,X49,X50,X51,X52,X53,X54),

cube(
X7,X4,X1,X8,X5,X2,X9,X6,X3,
X43,X11,X12,X44,X14,X15,X45,X17,X18,
X19,X20,X21,X22,X23,X24,X25,X26,X27,
X28,X29,X46,X31,X32,X47,X34,X35,X48,
X37,X38,X39,X40,X41,X42,X36,X33,X30,
X16,X13,X10,X49,X50,X51,X52,X53,X54)
    ).


% rotate "back" slice

mov(b,
cube(
X1,X2,X3,X4,X5,X6,X7,X8,X9,
X10,X11,X12,X13,X14,X15,X16,X17,X18,
X19,X20,X21,X22,X23,X24,X25,X26,X27,
X28,X29,X30,X31,X32,X33,X34,X35,X36,
X37,X38,X39,X40,X41,X42,X43,X44,X45,
X46,X47,X48,X49,X50,X51,X52,X53,X54),

cube(
X1,X2,X3,X4,X5,X6,X7,X8,X9,
X10,X11,X37,X13,X14,X38,X16,X17,X39,
X21,X24,X27,X20,X23,X26,X19,X22,X25,
X52,X29,X30,X53,X32,X33,X54,X35,X36,
X28,X31,X34,X40,X41,X42,X43,X44,X45,
X46,X47,X48,X49,X50,X51,X18,X15,X12)
    ).

% turn the cube around vertical axis

mov(t,
cube(
X1,X2,X3,X4,X5,X6,X7,X8,X9,
X10,X11,X12,X13,X14,X15,X16,X17,X18,
X19,X20,X21,X22,X23,X24,X25,X26,X27,
X28,X29,X30,X31,X32,X33,X34,X35,X36,
X37,X38,X39,X40,X41,X42,X43,X44,X45,
X46,X47,X48,X49,X50,X51,X52,X53,X54),

cube(
X10,X11,X12,X13,X14,X15,X16,X17,X18,
X19,X20,X21,X22,X23,X24,X25,X26,X27,
X28,X29,X30,X31,X32,X33,X34,X35,X36,
X1,X2,X3,X4,X5,X6,X7,X8,X9,
X43,X40,X37,X44,X41,X38,X45,X42,X39,
X48,X51,X54,X47,X50,X53,X46,X49,X52)
    ).


% ---------------------------------------------------------------
% some special "moves" that are mapped to longer move sequences
% ---------------------------------------------------------------

mov(sp1,X,Y) :-
move_sequence([-f, +d, +f, -d, -l, -d, +l], X, Y).

mov(sp2,X,Y) :-
move_sequence([+f, -d, -f, +d, -r, +d, +r], X, Y).

mov(sp3,X,Y) :-
move_sequence([-b, +r, -d, -r, +d, +b], X, Y).

mov(sp4,X,Y) :-
move_sequence([-b, -d, +r, +d, -r, +b], X, Y).

mov(sp5,X,Y) :-
move_sequence([-b, +r, -d, -r, +d, +b, +f, -d, -l, +d, +l, -f], X, Y).

mov(sp6,X,Y) :-
move_sequence([-l, -l, -u, -d, -d, +l, +r, +f, +f, -l, -r, +u, -l, -l],
X, Y).

mov(sp7,X,Y) :-
move_sequence([-l, -l, -u, +l, +r, +f, +f, -l, -r, -d, -d, +u, -l, -l],
X, Y).

mov(sp8,X,Y) :-
move_sequence([-l, -l, +u, +u, -b, -b, +u, +r, +r, +f, +f, +r, +r, +f,
+f, +r, +r, +f, +f, -u, -b, -b, +u, +u, -l, -l], X, Y).

mov(sp9,X,Y) :-
move_sequence([-r, -d, -l, +d, +r, -d, +l, +d], X, Y).

mov(sp10,X,Y) :-
move_sequence([-d, -l, +d, -r, -d, +l, +d, +r], X, Y).

mov(sp11,X,Y) :-
move_sequence([+l, -b, -b, +f, -l, -f, +l, +f, -l, -f, +l, +f, -l, -f,
+l, -b, -b, -l], X, Y).
```

```
mov(sp12,X,Y) :-
move_sequence([-b, +r, -d, -r, +d, +r, -d, -r, +d, +r, -d, -r, +d, +b],
X, Y).

mov(sp13,X,Y) :-
move_sequence([+f, +u, -f, -u, +f, +u, -f, -u], X, Y).

mov(sp14,X,Y) :-
move_sequence([+u, +f, -u, -f, +u, +f, -u, -f], X, Y).

% ---------------------------------------------------------------
% these helpers call the actual rotations
% ---------------------------------------------------------------

% clockwise move
move(+Move, OldState, NewState) :-
mov(Move, OldState, NewState).

% counterclockwise move
move(-Move, OldState, NewState) :-
mov(Move, NewState, OldState).

% empty move list
move_sequence([],X,X).

% longer sequence
move_sequence([Move|Othermoves], X, Z) :-
move(Move,X,Y),
move_sequence(Othermoves,Y,Z).

% -----------------------------------------------------------------------
% strategy: solve the cube in stages
% stages   1-4: put the upper slice edge pieces (two-colored) in place
% stages   5-8: put the upper slice corner pieces (three-colored) in place
% stage      9: turn the cube so that the 'F' face center and the upper
%               slice 'F' face pieces are on the same (front) face
% stages 10-20: put the middle slice corner pieces (two-colored) in place
% stages 21-24: form a one-color "cross" on the 'D' face with bottom slice
%               (two-colored) edge pieces
% stages 25-27: put the bottom slice edge pieces (two-colored) in place
% stage     28: put the bottom slice corner pieces in place
%
%
% here are the cube goal states for each stage
% -----------------------------------------------------------------------

get_goal(1,cube(
_,'F',_,_,_,_,_,_,_,
_,_,_,_,_,_,_,_,_,
_,_,_,_,_,_,_,_,_,
_,_,_,_,_,_,_,_,_,
_,_,_,_,_,_,_,'U',_,
_,_,_,_,_,_,_,_,_)
    ).

get_goal(2,cube(
_,'F',_,_,_,_,_,_,_,
_,'R',_,_,_,_,_,_,_,
_,_,_,_,_,_,_,_,_,
_,_,_,_,_,_,_,_,_,
_,_,_,_,_,'U',_,'U',_,
_,_,_,_,_,_,_,_,_)
    ).

get_goal(3,cube(
_,'F',_,_,_,_,_,_,_,
_,'R',_,_,_,_,_,_,_,
_,'B',_,_,_,_,_,_,_,
_,_,_,_,_,_,_,_,_,
_,'U',_,_,_,'U',_,'U',_,
_,_,_,_,_,_,_,_,_)
    ).

get_goal(4,cube(
_,'F',_,_,_,_,_,_,_,
_,'R',_,_,_,_,_,_,_,
_,'B',_,_,_,_,_,_,_,
_,'L',_,_,_,_,_,_,_,
_,'U',_,'U',_,'U',_,'U',_,
_,_,_,_,_,_,_,_,_)
    ).

get_goal(5,cube(
'F','F',_,_,_,_,_,_,_,
_,'R',_,_,_,_,_,_,_,
_,'B',_,_,_,_,_,_,_,
_,'L','L',_,_,_,_,_,_,
_,'U',_,'U',_,'U','U','U',_,
_,_,_,_,_,_,_,_,_)
    ).

get_goal(6,cube(
'F','F','F',_,_,_,_,_,_,
'R','R',_,_,_,_,_,_,_,
_,'B',_,_,_,_,_,_,_,
_,'L','L',_,_,_,_,_,_,
_,'U',_,'U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).
```

```
get_goal(7,cube(
'F','F','F',_,_,_,_,_,_,
'R','R','R',_,_,_,_,_,_,
'B','B',_,_,_,_,_,_,_,
_,'L','L',_,_,_,_,_,_,
_,'U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).

get_goal(8,cube(
'F','F','F',_,_,_,_,_,_,
'R','R','R',_,_,_,_,_,_,
'B','B','B',_,_,_,_,_,_,
'L','L','L',_,_,_,_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).

get_goal(9,cube(
'F','F','F',_,'F',_,_,_,_,
'R','R','R',_,_,_,_,_,_,
'B','B','B',_,_,_,_,_,_,
'L','L','L',_,_,_,_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).

get_goal(10,cube(
'F','F','F',_,'F',_,_,_,_,
'R','R','R',_,_,_,_,_,_,
'B','B','B',_,_,_,_,'L',_,
'L','L','L',_,_,_,_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,'F',_)
    ).

get_goal(11,cube(
'F','F','F','F','F',_,_,_,_,
'R','R','R',_,_,_,_,_,_,
'B','B','B',_,_,_,_,_,_,
'L','L','L',_,_,'L',_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).

get_goal(12,cube(
'R','R','R',_,'R',_,_,_,_,
'B','B','B',_,_,_,_,_,_,
'L','L','L',_,_,'L',_,_,_,
'F','F','F','F','F',_,_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).

get_goal(13,cube(
'R','R','R',_,'R',_,_,_,_,
'B','B','B',_,_,_,_,_,_,
'L','L','L',_,_,'L',_,'F',_,
'F','F','F','F','F',_,_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,'R',_)
    ).

get_goal(14,cube(
'R','R','R','R','R',_,_,_,_,
'B','B','B',_,_,_,_,_,_,
'L','L','L',_,_,'L',_,_,_,
'F','F','F','F','F','F',_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).

get_goal(15,cube(
'B','B','B',_,'B',_,_,_,_,
'L','L','L',_,_,'L',_,_,_,
'F','F','F','F','F','F',_,_,_,
'R','R','R','R','R',_,_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).

get_goal(16,cube(
'B','B','B',_,'B',_,_,_,_,
'L','L','L',_,_,'L',_,_,_,
'F','F','F','F','F','F',_,'R',_,
'R','R','R','R','R',_,_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,'B',_)
    ).

get_goal(17,cube(
'B','B','B','B','B',_,_,_,_,
'L','L','L',_,_,'L',_,_,_,
'F','F','F','F','F','F',_,_,_,
'R','R','R','R','R','R',_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).

get_goal(18,cube(
'L','L','L',_,'L','L',_,_,_,
'F','F','F','F','F','F',_,_,_,
'R','R','R','R','R','R',_,_,_,
'B','B','B','B','B',_,_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).

get_goal(19,cube(
'L','L','L',_,'L','L',_,_,_,
'F','F','F','F','F','F',_,_,_,
'R','R','R','R','R','R',_,'B',_,
'B','B','B','B','B',_,_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,'L',_)
    ).

get_goal(20,cube(
'L','L','L','L','L','L',_,_,_,
'F','F','F','F','F','F',_,_,_,
'R','R','R','R','R','R',_,_,_,
'B','B','B','B','B','B',_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,_,_,_,_,_,_,_,_)
    ).

get_goal(21,cube(
'L','L','L','L','L','L',_,_,_,
'F','F','F','F','F','F',_,_,_,
'R','R','R','R','R','R',_,_,_,
'B','B','B','B','B','B',_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,'D',_,_,_,_,_,_,_)
    ).

get_goal(22,cube(
'L','L','L','L','L','L',_,_,_,
'F','F','F','F','F','F',_,_,_,
'R','R','R','R','R','R',_,_,_,
'B','B','B','B','B','B',_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,'D',_,'D',_,_,_,_,_)
    ).

get_goal(23,cube(
'L','L','L','L','L','L',_,_,_,
'F','F','F','F','F','F',_,_,_,
'R','R','R','R','R','R',_,_,_,
'B','B','B','B','B','B',_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,'D',_,'D',_,'D',_,_,_)
    ).

get_goal(24,cube(
'L','L','L','L','L','L',_,_,_,
'F','F','F','F','F','F',_,_,_,
'R','R','R','R','R','R',_,_,_,
'B','B','B','B','B','B',_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,'D',_,'D',_,'D',_,'D',_)
    ).

get_goal(25,cube(
'L','L','L','L','L','L',_,'L',_,
'F','F','F','F','F','F',_,_,_,
'R','R','R','R','R','R',_,_,_,
'B','B','B','B','B','B',_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,'D',_,'D',_,'D',_,'D',_)
    ).

get_goal(26,cube(
'L','L','L','L','L','L',_,'L',_,
'F','F','F','F','F','F',_,'F',_,
'R','R','R','R','R','R',_,_,_,
'B','B','B','B','B','B',_,_,_,
'U','U','U','U',_,'U','U','U','U',
_,'D',_,'D',_,'D',_,'D',_)
    ).

get_goal(27,cube(
'L','L','L','L','L','L',_,'L',_,
'F','F','F','F','F','F',_,'F',_,
'R','R','R','R','R','R',_,'R',_,
'B','B','B','B','B','B',_,'B',_,
'U','U','U','U',_,'U','U','U','U',
_,'D',_,'D',_,'D',_,'D',_)
    ).

get_goal(28,cube(
'L','L','L','L','L','L','L','L','L',
'F','F','F','F','F','F','F','F','F',
'R','R','R','R','R','R','R','R','R',
'B','B','B','B','B','B','B','B','B',
'U','U','U','U',_,'U','U','U','U',
'D','D','D','D','D','D','D','D','D')
    ).

% --------------------------------------------------------------
% the search space is narrowed by listing all allowed rotations
% for each stage separately
% ----------------------------------------------------------------

cand(1, [r, d, f, t]).
cand(2, [r, d, f, t]).
cand(3, [r, d, f, t]).
cand(4, [r, d, f, t]).
cand(5, [f, r, d, t]).
cand(6, [r, d, f, t]).
cand(7, [r, d, f, t]).
cand(8, [r, d, f, t]).
cand(9, [u, t]).
cand(10, [f, d, r, sp1, sp2]).
cand(11, [sp1, sp2]).
cand(12, [t]).
cand(13, [f, d, r, sp1, sp2]).
cand(14, [sp1,sp2]).
cand(15, [t]).
cand(16, [f, d, r, sp1, sp2]).
cand(17, [sp1,sp2]).
cand(18, [t]).
cand(19, [f, d, r, sp1, sp2]).
cand(20, [sp1,sp2]).
cand(21, [t, sp3, sp4, sp5]).
cand(22, [t, sp3, sp4, sp5]).
cand(23, [t, sp3, sp4, sp5]).
cand(24, [t, sp3, sp4, sp5]).
cand(25, [t, u, sp6, sp7, sp8]).
cand(26, [t, u, sp6, sp7, sp8]).
cand(27, [t, u, sp6, sp7, sp8]).
cand(28, [t, u, sp9, sp10, sp11, sp12, sp13, sp14]).

% catch a candidate move from the list for each stage

get_candidate(Stage, Move) :-
cand(Stage, Movelist),
nth1(_,Movelist,Move).

% ----------------------------------------------------------------
% these helpers reject some unwise move candidates
% check whether the proposed move should be tried or not
% ----------------------------------------------------------------

% first, if this move is negation of previous move, we reject it

conflict([+Previous|_], -This) :- This=Previous.
conflict([-Previous|_], +This) :- This=Previous.

% also, maximum moves in the direction is 2 (cw) or 1 (ccw)
% (other can be reached with rotation to the other direction)
```

```
conflict([-Previous|_], -This) :- This=Previous.
conflict([+Previous|[+Previous|_]],+This) :- This=Previous.

% ------------------------------------------------------------
% rotate performs the BFS search from given cube state to get the
% the solution (a move list) for the defined stage.
% New moves are proposed by get_move (cw and ccw directions).
% ------------------------------------------------------------

% the list is empty, does not change the cube

rotate(_,[],State,State).

% Moves collects the answer move list (in REVERSE order).
% State is the current state of the cube
% Criteria contains the list of the pieces to be positioned in this stage

rotate(Stage, Moves, State, Stagegoal) :-
rotate(Stage, Priormoves, State, Nextstate),
get_move(Stage, Nextmove, Nextstate, Stagegoal),
not(conflict(Priormoves,Nextmove)),
append([Nextmove], Priormoves, Moves).

get_move(Stage, +Move, State, Stagegoal) :-
get_candidate(Stage, Move),
mov(Move, State, Stagegoal).

get_move(Stage, -Move, State, Stagegoal) :-
get_candidate(Stage, Move),
mov(Move, Stagegoal, State).

% -----------------------------------------------------------
% Here are the topmost predicates that collect the answer
% -----------------------------------------------------------

% updates the stage
get_stage(Stage, Newstage) :-
Newstage is Stage + 1.

% the cube should be finished after stage 28, so we stop here.
stage([],29,_).

% this builds the result move list by moving from stage to stage
% and joining the solutions gathered for each stage

stage(Movelist,Currentstage,Cubestate) :-
get_goal(Currentstage,Targetstate),
rotate(Currentstage,Stagemoves,Cubestate,Targetstate),
reverse(Stagemoves,Newmoves),
move_sequence(Newmoves,Cubestate,Newstate),
get_stage(Currentstage,Newstage),
stage(Newlist,Newstage,Newstate),
append(Newmoves,Newlist,Movelist).

% a little UI helper
solve(Movelist,Cube_state) :-
stage(Movelist, 1, Cube_state).
```

## 6 Example data and a test case

### 6.1 How to use the program

Import the program file ('rubik.prolog') using *consult*. Then, form a cube listing (using *cube*) and use it to make query with *solve*, which returns the answer.

### 6.2 Example

Here is an example of a query that returns the solution (move list) in X.

```
1 ?- solve(X,cube('F','R','U','B','B','L','U','L','R','B','D','U','U','L',
'B','D','L','L','L','R','D','U','F','B','F','R','B','B','L','U','R','R',
'D','L','U','R','R','D','F','F','U','F','R','U','L','B','D','F','F','D',
'B','D','F','D')).

X = [+d, +d, +r, -f, -f, -d, +r, +r, +f, +t, +r, +r, -t, +d, +f, -t, -f,
+t, -f, +f, -r, +f, +r, -f, +r, +t, +r, -d, -r, -f, -d, +f, -t, +r, +d,
+f, +d, +d, -f, -r, -t, -f, -r, +d, +r, +d, +f, +t, +u, +u, +t, +t, +sp1,
-d, +sp1, +t, -sp1, -d, +sp1, +t, -sp1, -d, +sp1, +t, +d, +sp1, +t, +sp5,
-t, +t, +sp4, -t, +sp7, +sp7, +sp8, +sp6, +sp8, +sp6, +sp12, +sp9, +sp12]

Yes
```

## 7 Summary

The logic program described in this document solves the Rubik's Cube, when given the current state of the cube as input. The solution is gathered in multiple stages with partial goals that put even more and more pieces in correct places.

The search made for the solution of a given stage is basically a plain brute-force attack with few optimizations: removing dummy moves and narrowing the move set depending on the stage.

The proram is not very efficient, but the performance is reasonable. Search optimizations and more innovative data structures (especially for the partial goals) are clear subjects for the further study. Also, cleaning the program output and extracting the special move sequences would also be a nice improvement.

## References

[1] Merrit, Dennis: "Prolog In Action: Solving Rubik's Cube",
<http://www.amzi.com/articles/rubik.htm>

[2] Jeays, Mark: "How to Solve Rubik's Cube 2.3",
<http://jeays.net/rubiks.htm>