CMSC 330: Organization of Programming Languages

Project 6 – Maze Solver & SAT in Prolog

Project 6

1. Implement recursive functions in Prolog
   • Some can be used to solve maze
2. Implement maze solver
3. Implement boolean formula evaluator

- Learn to use
  • Lists
  • Recursion
  • Backtracking
Part 1 – Recursion

- prod(L,R)
  - R = product of all ints in L
- fill(N,X,R)
  - R = list of N copies of X
- genN(N,R)
  - 0 ≤ R ≤ N-1, in order
- genXY(N,R)
  - R = [X,Y], 0 ≤ X,Y ≤ N-1 in order
- flat(L,R)
  - R = elements of L, concatenated in single list

Part 2 – Maze

- 2D square maze with walls separating cells
- Start & end located anywhere in maze
- Examples

| +---|---|---|---|---|--- | +---|---|---|---|---|--- |
|   e|   |   |   |   |   | e   |   |   |   |   |   |
| +   + + ++ | +   + ++ |
|   |   |   |   |   |   |   |   |   |   |   |   |
| +   + + ++ | +   + ++ |
|   |   |   |   |   |   |   |   |   |   |   |   |
| +   + + ++ | +   + ++ |
|   | e |   |   |   |   | s   |   |   |   |   |   |
| +---|---|---|---|---|--- | +---|---|---|---|---|--- |

Solvable

| +---|---|---|---|---|--- | +---|---|---|---|---|--- |
|   e|   |   |   |   |   | e   |   |   |   |   |   |
| +   + + ++ | +   + ++ |
|   |   |   |   |   |   |   |   |   |   |   |   |
| +   + + ++ | +   + ++ |
|   |   |   |   |   |   |   |   |   |   |   |   |
| +   + + ++ | +   + ++ |
|   | s |   |   |   |   | s   |   |   |   |   |   |
| +---|---|---|---|---|--- | +---|---|---|---|---|--- |

Unsolvable
**Part 2 – Maze Specification**

- **Maze**
  - An $n \times n$ array of cells
  - With specified starting & end cell
  - May include paths

- **Cells**
  - May have openings in walls
    - Specified as udlr → up, down, left, right
  - Each opening has a weight

- **Paths**
  - Sequence of directions (udlr) from starting cell

---

**Part 2 – Analyze & Process Maze**

- **Find maze properties**
  - Number of closed cells
  - Number of openings of each type (udlr)

- **Find all valid paths**
  - Path invalid if does not pass through opening
  - Path does not need to go from maze start to finish
  - Cost of path = sum of weight of each opening on path

- **List maze locations by distance from start**
  - $[[0,[[1,2]]],[1,[[1,1],[1,3]]],...]$

- **Decide whether maze is solvable**
Part 2 – Maze Specification in Prolog

- Maze given in database as 3 types of facts
  - Maze info (Size, StartX, StartY, EndX, EndY)
    - maze(4,0,3,0,0).
  - Cell info (Xcoord, Ycoord, OpeningList, WeightList)
    - cell(0,0,[d], [0.391538986557049]).
    - cell(1,0,[r,d],[16.597130417636, 0.889878639213553]).
  - Path info (PathName, StartX, StartY, DirectionList)
    - path("path1",0,3,[u,r,u,l,u]).
    - path("path2",0,3,[u,r,r,u,l,l,u]).

- Access maze information as queries
  - maze(X,_,_,_).
  - X = 4.

Part 3 – Boolean Formulae

- Representation
  - Booleans → t, f
  - Variables → x, y, z, etc…
  - Operations → no, and, or, every, exists
    - Represented as lists
      - [no, x], [and, x, y], [exists, x, x]
    - May be nested
      - [no, [and, x, [exists, x, x]]]
  - Variable assignments → lists of true variables
    - Variables not in assignment are false
Part 3 – Boolean Functions

Evaluation
- \( \text{eval}(F,A,R) \)
  - \( R \) is \( t \) if formula \( F \) is true with assignment \( A \), \( f \) otherwise

VarsOf
- \( \text{varsOf}(F,R) \)
  - \( R \) is list of free variables in \( F \) (in sorted order, no duplicates)

Satisfiability
- \( \text{sat}(F,R) \)
  - \( R \) is assignment that satisfies \( F \)
  - \( R = [ ] \) if all assignments satisfy \( F \)
  - \( R = [ ] \) if all false variables satisfy \( F \) (unfortunately same)

Project Files

Your code
- logic.pl

Public tests
- publicRecursion1.pl, publicRecursion2.pl
- publicMaze1.pl, publicMaze2.pl
- publicEval.pl, publicVarsOf.pl, publicSat.pl

Utility files
- goTest.pl - test driver for all public tests
Public Tests

- Contain following test goals
  - public_fill
  - public_genN
  - public_genXY
  - public_flat
  - public_stats
  - public_validPath
  - public_findDistance
  - public_solve
  - public_eval, public_varsOf, public_sat

Differences From Project 1 & 3

- Utilize backtracking in Prolog
  - Return single answer, instead of list of answers
  - Request additional answers by typing ;

- Tests will collect all answers, using Prolog utils
  - Findall – list of all answers
  - Setof – order list of all answers (duplicates removed)
Using Prolog From Terminal / Shell

- Go to directory p6 (from p6.zip) containing proj files
  - E.g., cd c:Users\myname\Desktop\p6
- Start Prolog interpreter
  - Type `swipl`
- Load Prolog code
  - Type `[‘logic.pl’]`.
  - Type `[‘publicRecursion1.pl’]`.
- Run public tests
  - Type `public_fill`
  - Etc...

Using Prolog From Terminal / Shell

- To run all public tests
  - Type `[‘goTest.pl’]`.
  - Type `run`.

- To modify your code
  - Edit `logic.pl`
  - Type `make`
Using Prolog From Windows

- Open folder p6 (from p6.zip) containing project files
- Start Prolog interpreter
  - Right click goTest.pl, select Open With -> swipl-win.exe
  - Terminal window opens running Prolog interpreter
    - goTest.pl loaded at same time
- To run all public tests
  - Type `run`.
- To modify your code
  - Edit logic.pl
  - Type `make`.