Prolog supervision work

Michaelmas 2010
David Eyers <David.Eyers@cl.cam.ac.uk>
Original author Dr Andrew Rice

1 Introduction

These questions form the suggested supervision work for the Prolog course. All students should attempt the basic questions. Ideally, students should ensure that they have access to a Prolog enviornment, and test their work within it. Those questions marked with an asterisk are more difficult although all students should be able to answer them with the help of their supervisor. Questions marked with a double asterisk are particularly challenging and are beyond the level required for producing a good answer in the exam.

Prolog contains a number of features and facilities not covered in the lectures such as: `assert`, `findall` and `retract`. Students should limit themselves to using only the features covered in the lecture course and are not expected to know about anything further. All questions can be successfully answered using only the lectured features.

Students are encouraged to contact me by email with bug-reports and solutions to double-asterisk questions.

2 Lecture 1

2.1 Unification

1. Unify these two terms by hand:
   - `tree(tree(tree(1,2),A,B),tree(C,tree(E,F,G)))`
   - `tree(C,tree(Z,C))`
2. Explain Prolog's behaviour when you unify `a(A)` with `A`.
3. Relate unification with ML type inference

2.2 The Zebra Puzzle

1. Implement and test the Zebra Puzzle solution
2. Explain how Clue 2 has been expressed in the Zebra Puzzle query

3 Lecture 2

3.1 Encoding arithmetic in Prolog

The `is` operator in Prolog evaluates arithmetic expressions. This built-in functionality can also be modelled within Prolog's logical framework.

Let the atom `i` represent the identity (1) and the compound term `s(A)` represent the successor of `A`. For example `4 = s(s(s(i)))`

Implement and test the following rules (note: you should not use `is` to do all of the arithmetical work):

1. `prim(A,B)` which is true if `A` is a number and `B` is its primitive representation
2. `plus(A,B,C)` which is true if `C is A+B` (all with primitive representations, `A` and `B` are both ground terms)
3. `mult(A,B,C)` which is true if `C is A*B` (all with primitive representations, `A` and `B` are both ground terms)

The development of arithmetic (and general computation) from first principles is considered more formally in the Foundations of Functional Programming course.

3.2 List Operations

1. Explain the operation of the `append/3` clause.

```
append([],A,A).
append([H|T],A,[H|R]) :- append(T,A,R).
```

2. Draw the Prolog search tree for `perm([1,2,3],A).`
3. Implement a clause `choose(N,L,R,S)` that chooses `N` items from `L` and puts them in `R` with the remaining elements in `L` left in `S`.
4. * What it the purpose of the following clauses:

```
a([H|T]) :- a([H|T],H).
a([],_).
a([H|T],Prev) :- H >= Prev, a(T,H).
```

5. * What does the following do and how does it work?:

```
b(X,X) :- a(X).
b(X,Y) :- append(A,[H1,H2|B],X), H1 > H2, append(A,[H2,H1|B],X1),
          b(X1,Y).
```

3.3 Generate and Test

The description of these problems will be given in the lecture.

1. Complete the Dutch National Flag solution
2. * Complete the 8-Queens solution
3. * Generalise 8-Queens to n-Queens
4. Complete the Anagram generator. In what situations is it more efficient to Test-and-Generate rather than Generate-and-Test?

4 Lecture 3

4.1 Symbolic Evaluation

1. Explain what happens when you put the clauses of the symbolic evaluator in a different order
2. Add additional clauses to the symbolic evaluator for subtraction and integer division (this is the `//` operator in Prolog i.e. `2 is 6//3`)

4.2 Negation

State and explain Prolog's response to the following queries:

1. `X=1.`
2. `not(X=1).`
3. `not(not(X=1)).`
4. `not(not(not(X=1))).`

In those cases where Prolog says 'yes' your answer should include the unified result for `X`.

4.3 Databases

We can use facts entered into Prolog as a general database for storing and querying information. This question considers the construction of a database containing information about students, their colleges and their grades in the various parts of the CS Tripos.

Each fact in our Prolog database corresponds to a row in a table of data. A table is constructed from rows produced by facts with the same name. The initial database of facts is as follows:

```
tName(dme26,'David Eyers').
tName(awm22,'Andrew Moore').

tCollege(dme26, 'King''s').
tCollege(awm22, 'Corpus Christi').

tGrade(dme26,'IA',2.1).
tGrade(dme26,'IB',1).
tGrade(dme26,'II',1).
tGrade(awm22,'IA',2.1).
tGrade(awm22,'IB',1).
tGrade(awm22,'II',1).
```

As an example, this database contains a table called 'tName' which contains two rows of two columns. The first column is the CRSID of the individual and the second column is their full name.

4.3.1 Part 1

1. Add your own details to the database.
2. Add a new table tDOB that contains CRSID and DOB.
3. Alter the database such that for some users their college is not present (this final step is necessary for testing your answers to the questions in Part 2)

4.3.2 Part 2

The next task is to provide rules and show queries that implement various queries of the database. You should answer each question with the Prolog facts and rules required to implement the query and also an example invocation of those rules.

For example:

```
% The full name of each person in the database
qFullName(A) :- tName(_,A).

% Example invocation
% qFullName(A).
```

Each query should return one row of the answer at a time, subsequent rows should be returned by backtracking.

For the example above:

```
?- qFullName(A).

A = 'David Eyers' ;

A = 'Andrew Moore'

Yes
```

- The descriptions that follow provide a plain English description of the query that you should implement, followed by the same query in SQL.
- SQL (Structured Query Language) is the industry standard language currently used to query relational databases—you will see more on this in the Databases course.
- The '?' notation in the SQL statements derives from the use of prepared statements in relational databases where (for efficiency) a single statement is sent to the database server and repeatedly evaluated with different values replacing the '?'. Interested students can consult the Java PreparedStatement documentation.

1. Full name and College attended.
   SELECT name,college FROM tName, tCollege WHERE tName.crsid = tCollege.crsid
2. Full name and College attended only including entries where the user can choose a single CRSID to include in the results.
   SELECT name,college FROM tName, tCollege WHERE tName.crsid = tCollege.crsid AND tName.crsid = ?
3. Full name and College attended or blank if the college is unknown.
   SELECT name,college FROM tName LEFT OUTER JOIN tCollege ON tName.crsid = tCollege.crsid
4. Full name and College attended. The full name or the college should be blank if either is unknown.
   SELECT name,college FROM tName FULL OUTER JOIN tDOB ON tName.crsid = tCollege.crsid
5. * Find the lowest grade where the CRSID is specified by the user. Note that this predicate should only return one result even when backtracking.
   SELECT min(grade) FROM tGrade WHERE crsid = ?

6. ** Find the number of people with a First class mark
   SELECT count(grade) FROM tGrade WHERE grade = 1

7. ** Find the number of First class marks awarded to each person. Your output should consist of a tuple (CRSID,NumFirsts) that iterates through all CRSIDs which have at least one First class mark upon backtracking
   SELECT crsid,count(grade) FROM tGrade WHERE grade=1 GROUP BY crsid
   Hint: This is not the number of rows with First class marks in the tGrade table. You will need build a list of First class CRSIDs by repeatedly querying tGrade and checking if the result is already in your list. Every time you find a new unique CRSID, increment an accumulator which will form the result.

5 Lecture 4

5.1 Countdown Numbers Game

1. Type in the example code which finds exact solutions from the lectures and test it;
2. Implement the predicate `range(Min,Max,Val)` which unifies `Val` with `Min` on the first evaluation, and then all values up to `Max-1` when backtracking;
3. Get the iterative deepening version of the numbers game working.

5.2 Graph searching

Implement search-based solutions for:

1. Missionaries and Cannibals: there are three missionaries, three cannibals and who need to cross a river. They have one boat which can hold at most two people. If, at any point, the cannibals outnumber the missionaries then they will eat them. Discover the procedure for a safe crossing.
2. * Towers of Hanoi: you have N rings of increasing size and three pegs. Initially the three rings are stacked in order of decreasing size on the first peg. You can move them between pegs but you must never stack a big ring onto a smaller one. What is the sequence of moves to move from all the rings from the first to the the third peg.
3. * Umbrella: A group of 4 people, Andy, Brenda, Carl, and Dana, arrive in a car near a friend's house, who is having a large party. It is raining heavily, and the group was forced to park around the block from the house because of the lack of available parking spaces due to the large number of people at the party. The group has only 1 umbrella, and agrees to share it by having Andy, the fastest, walk with each person into the house, and then return each time. It takes Andy 1 minute to walk each way, 2 minutes for Brenda, 5 minutes for Carl, and 10 minutes for Dana. It thus appears that it will take a total of 19 minutes to get everyone into the house. However, Dana indicates that everyone can get into the house in 17 minutes by a different method. How? The individuals must use the umbrella to get to and from the house, and only 2 people can go at a time (and no funny stuff like riding on someone's back, throwing the umbrella, etc.). (This puzzle included with kind permission from `http://www.puzz.com/`)

6 Lecture 5

6.1 Sorting

Implement the following sorting algorithms in Prolog:

1. Finding the minimum element from the list and recursively sorting the remainder.
2. Quicksort.
3. * Quicksort where the partitioning step divides the list into three groups: those items less than the pivot, those items equal to the pivot and those items greater than the pivot. Explain in what situations this additional complexity might be desirable.
4. * Quicksort with the append removed using difference lists. Note: you do not need to alter the partitioning clauses.
5. * Mergesort

6.2 Towers of Hanoi

The Towers of Hanoi problem can be solved without requiring an inefficient graph search. Discover the algorithm required to do this from the lecture notes or the web and implement it. Once you have a simple list-based implementation rewrite it to use difference lists.

6.3 Dutch Flag

Earlier in the course we solved the dutch flag problem using generate and test. A more efficient approach is to make one pass through the initial list collecting three separate lists (one for each colour). When you reach the end of the initial list you append the three separate collection lists together and you are done. Implement this algorithm using normal lists and then rewrite it to use difference lists.

7 Lecture 6

7.1 Sudoku Solver

Extend the CLP-based 2x2 Sudoku solver given in the lecture to a 3x3 grid and test it.

7.2 Cryptarithmetic

Here is a classic example of a cryptarithmetic puzzle:

```
    S E N D
+   M O R E
-----------
  M O N E Y
```

The problem is to find an assignment of the numbers 0–9 (inclusive) to the letters S,E,N,D,M,O,R,E,Y such that the arithmetic expression holds and the numeric value assigned to each letter is unique.

We can formulate this problem in CLP as follows:

```
solve1([S,E,N,D],[M,O,R,E],[M,O,N,E,Y]) :-
  Var = [S,E,N,D,M,O,R,Y],
  Var in 0..9, all_different(Var),
            1000*S + 100*E + 10*N + D +
            1000*M + 100*O + 10*R + E #=
  10000*M + 1000*O + 100*N + 10*E + Y,
  labeling([],Var).
```

1. Get the example above working in your Prolog interpreter. How many unique solutions are there?
2. A further requirement of these types of puzzle is that the leading digit of each number in the equation is not zero. Extend your program to incorporate this. The CLP operator for arithmetic not-equals is `#\=`. How many unique solutions remain now?
3. * Extend your program to work in an arbitrary base (rather than base 10), the domain of your variables should change to reflect this. How many solutions of the puzzle above are there in base 16?
4. * Consult the Prolog documentation regarding the findall predicate. Use findall within a predicate `count(Base,N)` that unifies `N` with the number of solutions in base `Base`.
5. * Use the `range/3` predicate from Lecture 4 to extend your count predicate to try all values from 1 to 50 as the base.
6. * Plot a graph of the number of solutions against the chosen base.

7.3 **Findall

The findall predicate is an extra-logical predicate for backtracking and collecting the results into a list. The implementation within Prolog is something along the lines of:

```
findall(Template,Goal,Solutions) :- call(Goal),
                                    assertz(findallsol(Template)),
                                    fail.
findall(Template,Goal,Solutions) :- collect(Solutions).

collect([Template|RestSols]) :- retract(findallsol(Template)),
                                !,
                                collect(RestSols).
collect([]).
```

1. ** Consult the Prolog documentation and work out what the above is doing. The predicate `assertz` adds a new clause to the end of the running Prolog program (i.e. the Prolog database) and the predicate retract removes a clause that unifies with its argument.
2. ** Develop an alternative to `findall` that does not use extra-logical predicates such as `assertz` and `retract`. This alternative will necessarily take the form of a pattern which you can apply to your clause to find all the possible results. The algorithm for the alternative `findall` proceeds as follows: maintain a list of solutions found so far, evaluate the target clause and repeatedly backtrack through it until it returns a value not in your list of results, add the result to the list and repeat.
3. ** Comment on the runtime complexity of the alternate `findall` compared to the built-in version.