Prolog Supervision Work

Andrew Rice[0],
Department of Computer Science and Technology,
Cambridge University.
acr31@cam.ac.uk

Academic Year 2019–20

Introduction

These questions form the suggested supervision material for the Prolog course. Supervisors are encouraged to select an appropriate subset of these for their students to complete.

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.

Questions are classified into the following types:

Bookwork questions that require the students to review the lectured material and locate the relevant information

Shallow questions that require recall of lectured material and its direct application in a formulaic manner

Deeper questions that require students to apply the lectured material in a new context or to relate material to each other but still with a clear right or wrong answer

Open ended questions requiring students to form their own viewpoints with various ways to interpret the answer

Prolog Basics

Review

(1R.1) Bookwork Specify the rules Prolog uses for unification

(1R.2) Shallow Unify these two terms by hand explaining which of the unification rules are invoked at which point:

```
tree(tree(tree(1,2),A,B),tree(C,tree(E,F,G)))
```

and

```
tree(C,tree(Z,C))
```

[0]Grateful thanks to Nik Sultana who lectured this course in 2016 and contributed to this question set.

Supervision work

(1S.1) Deeper Different implementations of Prolog produce different behaviour when you attempt to unify `a(A)` with `A` . Describe the various possibilities which might arise.

(1S.2) Open How does unification relate to ML type inference? What is the ML equivalent of unifying `a(A)` with `A` ? What behaviour is desirable in this case?

Zebra Puzzle

Review

(2R.1) Shallow Implement and test the Zebra Puzzle solution

(2R.2) Shallow Explain how Clue 2 has been expressed in the zebra query

Supervision work

(2S.1) Open What are the characteristics of this puzzle that make it amenable to our solution method? Come up with another puzzle which can be solved with the same approach.

Rules

Review

(3R.1) Bookwork Build a glossary of the important Prolog terminology so far

(3R.2) Shallow What are the FOL formula for the `valuable` rules in the slides? How would you join the rules together in FOL?

Supervision work

(3S.1) Deeper Use rules to rewrite the Zebra puzzle solution. How short can you make it without adding more terms to the query?

Lists

Review

(4R.1) Bookwork Summarise Prolog's List syntax

(4R.2) Bookwork Drawing a search tree is a useful exercise because it helps to capture the search algorithm itself. Draw out the search tree for last([1,2],A).

Supervision work

(4S.1) Shallow Consider the implementation of append given below. Explain how it should be used and draw out a search tree for a representative example.

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

(4S.2) Deep You are given two implementations of member

member(H,[H|_]).
member(H,[_|T]) :- **member**(H,T).

% or, alternatively

member(X,Y) :- **append**(_,[X|_],Y).

1. Discuss how the second implementation compares with the use of 'partial application' from functional programming, and how it differs.

2. If the two implementations are equivalent, should a programmer simply inline all calls to member with a call to append?

3. If the two implementations are equivalent, what advantages does one form of expression have over the other?

4. Prove (informally) why the two are logically equivalent.

(4S.3) Deeper What it the purpose of the following clauses:

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

(4S.4) Deeper 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).

Arithmetic

Review

(5R.1) Bookwork What's the difference between `A = 1+2` and `A is 1+2` ?

(5R.2) Bookwork What is Last Call Optimisation?

(5R.3) Bookwork Try the 'test to destruction' (see the slides for what this means) on the len predicate

Supervision work

(5S.1) Deeper The `is` operator in Prolog evaluates arithmetic expressions. This builtin functionality can also be modelled within Prolog's logical framework. Let the atom `z` represent the smallest number (this is zero) and the compound term `s(A)` represent the successor of A. For example `3 = s(s(s(z)))` . Implement and test the following rules (note: you should not use `is` to do all of the arithmetical work):

`prim(A,B)` which is true if A is a number and B is its primitive representation

`plus(A,B,C)` which is true if C is A+B (all with primitive representations, A and B are both ground terms

`mult(A,B,C)` which is true if C is A*B (all with primitive representations, A and B are both ground terms)

(5S.2) Shallow Once you have implemented the `prim` predicate described above, give an instance for which it can be reversed. Can you find an instance for which it cannot be reversed?

(5S.3) Shallow Write a reversible version of `prim` . (Hint: look up the Prolog primitives `var/1` and `integer/1` .)

(5S.4) Shallow Explain what each of these queries evaluate to, and explain why you got the evaluation you did.

1. `1 + 1 = 2`
2. `1 + 1 is 2`
3. `1 + 1 =:= 2`
4. `One + One = Two`
5. `prim(1, One), prim(2, Two), plus(One, One, Two)`

(This requires you to have implemented the `prim` and `plus` predicates described earlier.)

(5S.5) Open If we ask Prolog to solve algebra then the `is` operator throws an error. For example the query `3 is A+2` fails. Give an example of how this is not the case with our `prim` and `plus` rules and explain what the difference is.

(5S.6) Deeper Come up with another example of a predicate which can be written with and without Last Call Optimisation. Create a test to demonstrate it.

Backtracking

Supervision work

These questions ask you to consider the use of predicates 'backwards' - i.e. swapping the inputs and outputs. Its easy to find out what they do by just trying it in the interpreter. However, the real question is to see whether you can reason it out from your knowledge of how the search and backtracking algorithms work.

(6S.1) Shallow What happens if the accumulator version of len is used 'backwards'?

(6S.2) Shallow What happens if take is used 'backwards'?

(6S.3) Shallow What happens if append is used 'backwards'?

Generate and Test

Review

(7R.1) Shallow Draw the Prolog search tree for perm([1,2],A).

(7R.2) Shallow Complete the Dutch National Flag solution

(7R.3) Deeper Complete the 8-Queens solution

Supervision work

(7S.1) Deeper Generalise 8-Queens to n-Queens

(7S.2) Deeper Complete the Anagram generator.

(7S.3) Deeper Extend the Anagram generator to look for pairs of words rather than a single word. For example, a pair anagram of the word HOTDOG would be the words HOT and DOG, or HOG and DOT.

(7S.4) Open In what situations is it more efficient to Test-and-Generate rather than Generate-and-Test?

Symbolic Evaluation

Review

(8R.1) Deeper Explain what happens when you put the clauses of the symbolic evaluator in a different order

Cut

Review

(9R.1) Bookwork Describe which choice points the cut operator removes

(9R.2) Shallow Review the example of the predicate whose logical interpretation is changed by the use of cut. Come up with your own example.

Negation

Review

(10R.1) Bookwork State the closed world assumption

(10R.2) Bookwork Explain in general how the use of negation in a program can cause it to have a non-logical behaviour

Supervision work

(10S.1) Shallow State and explain Prolog's response to the following queries:

```
X=1.
not(X=1).
not(not(X=1)).
not(not(not(X=1))).
```

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

Databases

Review

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(acr31,'Andrew Rice').
tName(arb33,'Alastair Beresford').
```

```
tCollege(acr31, 'Churchill').
tCollege(arb33, 'Robinson').

tGrade(acr31,'IA',2.1).
tGrade(acr31,'IB',1).
tGrade(acr31,'II',1).
tGrade(arb33,'IA',2.1).
tGrade(arb33,'IB',1).
tGrade(arb33,'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.

(11R.1) Bookwork Add your own details to the database.

(11R.2) Bookwork Add a new table tDOB which contains CRSID and DOB.

(11R.3) Bookwork 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)

Supervision work

The next task is to provide rules and show queries which implement various queries of the database. By answering these questions you are demonstrating that you understand how backtracking works and how to control it. You should answer each question with the Prolog facts and rules required to implement the query and also an example invocation of these rules. For example:

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

A = 'Alastair Beresford'

Yes
```

The descriptions that follow provide a plain English description of the query you should implement followed by the same query in SQL. SQL (Structured Query Language) is the industry standard language 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 or wait for the Further Java course.

(11S.1) Bookwork Full name and College attended.

```
SELECT name,college from tName, tCollege
WHERE tName.crsid = tCollege.crsid
```

(11S.2) Bookwork 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 = ?
```

(11S.3) Shallow 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
```

(11S.4) Shallow 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 tCollege
ON tName.crsid = tCollege.crsid
```

(11S.5) Deeper Find the lowest (numerically) 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 = ?
```

(11S.6) Deeper Find the number of First class marks given out

```
SELECT count(grade) from tGrade
WHERE grade = 1
```

(11S.7) Deeper (Hard) Find the number of First class marks awarded to each person. Your output should consist of a tuple (CRSID,NumFirsts) which 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

Countdown

Review

(12R.1) Bookwork What is iterative deepening

Supervision work

(12S.1) Deeper Implement a clause choose(N,L,R,S) which chooses N items from L and puts them in R with the remaining elements in L left in S

(12S.2) Shallow Add additional clauses to the symbolic evaluator for subtraction and integer division (this is the // operator in Prolog i.e. 2 is 6//3) and then get the countdown game working

Graph search

Review

(13R.1) Bookwork Give the standard Prolog patterns for graph search programs: search, search when there are cycles, search with a log, search when there are cycles with a log

(13R.2) Bookwork Why is it important to minimize redundancy in your state representation?

Supervision work

(13S.1) Shallow Missionaries and Cannibals: there are three missionaries and three cannibals 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.

(13S.2) Deeper Towers of Hanoi: you have 3 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 all the rings from the first to the the third peg.

(13S.3) Deeper 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/)

Difference lists

Review

(14R.1) Bookwork What is a difference list? How does one write an empty difference list? Why?

(14R.2) Bookwork Derive the clause for appending two difference lists and explain why it works

Supervision work

Implement the following sorting algorithms in Prolog

(14S.1) Deeper Finding the minimum element from the list and recursively sorting the remainder.

(14S.2) Deeper Quicksort

(14S.3) Deeper 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.

(14S.4) Deeper In what situations might this three way pivot be desirable.

(14S.5) Deeper Quicksort with the append removed (by using difference lists).

(14S.6) Deeper Mergesort

(14S.7) Deeper 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.

(14S.8) Deeper 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.

Sudoku

Review

(15R.1) Bookwork Why is the `range` based solution so much slower than the `perm` based solution? What's the general principle here?

Constraints

Review

(16R.1) Shallow Extend the CLP-based 2x2 Sudoku solver given in the lecture to a 3x3 grid and test it.

Supervision work

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 ins 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).
```

(16S.1) Shallow Get the example above working in your Prolog interpreter. How many distinct solutions are there?

(16S.2) Deeper 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 distinct solutions remain now?

(16S.3) Deeper 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?

Extra-fun

Supervision work

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([]).

(17S.1) Deeper 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 and the predicate retract removes a clause which unifies with its argument.