COMP3411/9414/9814 Artificial Intelligence

Session 1, 2018

Assignment 1 - Prolog Programming

Due: Friday 6 April, 11:59pm
Marks: 12% of final assessment for COMP3411/9414/9814 Artificial Intelligence

---

Specification

In this assignment, you are to write Prolog procedures to perform some list and tree operations. The aim of the assignment is to give you experience with typical Prolog programming techniques.

At the start of your program, place a comment containing **your full name, student number and assignment name**. You may add additional information like the date the program was completed, etc. if you wish.

At the start of each Prolog predicate that you write, write a comment describing the overall function of the predicate.

Advice on the use of comments and meaningful identifiers in Prolog can be found under comments in the Prolog Dictionary.

Testing Your Code

A significant part of completing this assignment will be testing the code you write to make sure that it works correctly. To do this, you will need to design test cases that exercise every part of the code.

You should pay particular attention to "boundary cases", that is, what happens when the data you are testing with is very small, or in some way special. For example:

- What happens when the list you input has no members, or only one member?
- Does you code work for lists with both even and odd numbers of members?
- Does your code work for negative numbers?

Note: not all of these matter in all cases, so for example with `sqrt_table`, negative numbers don't have square roots, so it doesn't make sense to ask whether your code works with negative numbers.

With each question, some example test data are provided to clarify what the code is intended to do. You need to design *further* test data. Testing, and designing test cases, is part of the total programming task.

It is important to use exactly the names given below for your predicates, otherwise the automated testing procedure will not be able to find your predicates and you will lose

marks. Even the capitalisation of your predicate names must be as given below.

---

1. Write a predicate `sumsq_neg(Numbers, Sum)` that sums the squares of only the negative numbers in a list of numbers. *Example:*

```
?- sumsq_neg([1,-3,-5,2,6,8,-2], Sum).
Sum = 38
```

This example computes `(-3)*(-3) + (-5)*(-5) + (-2)*(-2)`. Think carefully about how the predicate should behave on the empty list — should it fail or is there a reasonable value that `Sum` can be bound to?

---

2. For the purposes of the examples in this question, assume that the following facts have been loaded into Prolog:

```
likes(mary, apple).
likes(mary, pear).
likes(mary, grapes).
likes(tim, mango).
likes(tim, apple).
likes(jane, apple).
likes(jane, mango).
```

NOTE: do **not** include these in your solution file.

Write a predicate `all_like_all(Who_List, What_List)` that takes a list of people `Who_List` and a list of items `What_List` and succeeds if every person in `Who_List` likes every item in `What_List`, according to the predicate `likes(Who, What)`. Your predicate should also succeed if either `Who_List` or `What_List` is empty. *Examples:*

```
?- all_like_all([jane,tim],[apple,mango]).
true

?- all_like_all([mary,tim],[apple,grapes]).
false.

?- all_like_all([],[bananas]).
true
```

Note that your `all_like_all` predicate will be tested with different `likes(Who, What)` facts to those in the examples.

---

3. Write a predicate `sqrt_table(N, M, Result)` that binds `Result` to the list of pairs consisting of a number and its square root, from `N` down to `M`, where `N` and `M` are non-negative integers, and `N >= M`. For example:

```
sqrt_table(7, 4, Result).
Result = [[7, 2.6457513110645907], [6, 2.449489742783178], [5, 2.23606797749979], [4, 2.0]]

?- sqrt_table(7, 8, Result).
false.
```

Note that the Prolog built-in function sqrt computes square roots, and needs to be evaluated using `is` to actually compute the square root:

```
?- X is sqrt(2).
X = 1.4142135623730951.
```

```
?- X = sqrt(2).
X = sqrt(2).
```

---

4. Write a predicate `chop_up(List, NewList)` that takes `List` and binds `NewList` to `List` with all sequences of *successive increasing* whole numbers replaced by a two-item list containing only the first and last number in the sequence. An example of successive increasing whole numbers is: `19,20,21,22`. (Note that the numbers have to be *successive* in the sense of increasing by exactly 1 at each step.) For example:

   ```
   ?- chop_up([9,10,5,6,7,3,1], Result).
   Result = [[9, 10], [5, 7], 3, 1]

   ?- chop_up([1,3,2,3,4], Result).
   Result = [1, 3, [2, 4]]
   ```

   In this example, the sequence `9,10` has been replaced by `[9,10]`, the sequence `5,6,7` has been replaced by `[5, 7]`, and `2,3,4` has been replaced by `[2, 4]`.

---

5. For this question we consider binary expression-trees whose leaves are either of the form `tree(empty, Num, empty)` where `Num` is a number, or `tree(empty, z, empty)` in which case we will think of the letter `z` as a kind of "variable". Every tree is either a leaf or of the form `tree(L, Op, R)` where `L` and `R` are the left and right subtrees, and `Op` is one of the arithmetic operators `'+'`, `'-'`, `'*'`, `'/'` (signifying addition, subtraction, multiplication and division).

   Write a predicate `tree_eval(Value, Tree, Eval)` that binds `Eval` to the result of evaluating the expression-tree `Tree`, with the variable `z` set equal to the specified `Value`. For example:

   ```
   ?- tree_eval(2, tree(tree(empty,z,empty),
                   '+',tree(tree(empty,1,empty),
                        '/',tree(empty,z,empty))), Eval).
   Eval = 2.5

   ?- tree_eval(5, tree(tree(empty,z,empty),
                   '+',tree(tree(empty,1,empty),
                        '/',tree(empty,z,empty))), Eval).
   Eval = 5.2
   ```

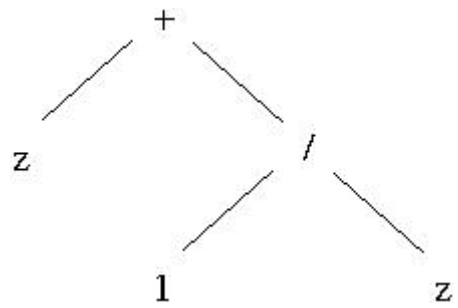

Illustration of the tree used in the example above.

Testing

This assignment will be marked on functionality in the first instance. However, you should always adhere to good programming practices in regard to structure, style and comments,

as described in the Prolog Dictionary. Submissions which score very low in the automarking will be examined by a human marker, and may be awarded a few marks, provided the code is readable.

Your code must work under the version of SWI Prolog used on the Linux machines in the UNSW School of Computer Science and Engineering. If you develop your code on any other platform, it is your responsibility to re-test and, if necessary, correct your code when you transfer it to a CSE Linux machine prior to submission.

Your code will be run on a few simple tests when you submit. So, it is a good idea to submit early and often so that potential problems with your code can be detected early. You will be notified at submission time if your code produces any compiler warnings. Please ensure that your final submission does not produce any such warnings (otherwise, marks will be deducted).

Submitting your assignment

Put the Prolog code for all problems into a single file for submission purposes.

COMP3411 students: to hand in, log in to a School of CSE Linux workstation or server, make sure that your program is in the current working directory, and use the Unix command:

```
% give cs3411 prolog mycode.pl
```

where *mycode.pl* is replaced by the name of the file with your code in it.

COMP9414/9814 students: to hand in, log in to a School of CSE Linux workstation or server, make sure that your program is in the current working directory, and use the Unix command:

```
% give cs9414 prolog mycode.pl
```

where *mycode.pl* is replaced by the name of the file with your code in it.

Please make sure your code works on CSE's Linux machines and generates no warnings. Remove all test code from your submission, including that for question 2. Make sure you have named your predicates correctly.

You can submit as many times as you like - later submissions will overwrite earlier ones. You can check that your submission has been received by using one of these commands:

```
% 3411 classrun -check prolog
% 9414 classrun -check prolog
```

The submission deadline is Friday 6 April, 11:59 pm.
15% penalty will be applied to the (maximum) mark for every 24 hours late after the deadline.

Questions relating to the project can be posted to the Forums on the course Web site.

If you have a question that has not already been answered on the Forum, you can email it to blair@cse.unsw.edu.au

Plagiarism Policy

Group submissions will not be allowed. Your program must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for any similar projects from previous years) and serious penalties will be applied, particularly in the case of repeat offences.

DO NOT COPY FROM OTHERS; DO NOT ALLOW ANYONE TO SEE YOUR CODE

Please refer to the UNSW Policy on Academic Honesty and Plagiarism if you require further clarification on this matter.

Good luck!
