COMP3411/9814 Artificial Intelligence

Summer session, 2020

Assignment 1 - Prolog Programming

Due: Monday 27 January, 11:59pm
Marks: 20% of final assessment for COMP3411/9814 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_even(Numbers, Sum)` that sums the squares of only the even numbers in a list of integers. *Example:*

```
?- sumsq_even([1,3,5,2,-4,6,8,-7], Sum).

Sum = 120 ;

false.
```

Note that it is the element of the list, not its position, that should be tested for oddness. (The example computes 2*2 + (-4)*(-4) + 6*6 + 8*8.) 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?

In order to decide whether a number is even or odd, you can use the built-in Prolog operator `N mod M`, which computes the remainder after dividing the whole number `N` by the whole number `M`. Thus a number `N` is even if the goal `0 is N mod 2` succeeds. Remember that arithmetic expressions like `X + 1` and `N mod M` are only evaluated, in Prolog, if they appear *after* the `is` operator. So `0 is N mod 2` works, but `N mod 2 is 0` doesn't work.

---

2. Suppose that a set of family relationships have been loaded into Prolog, for exmaple:

```
parent(jim, brian).
parent(brian, jenny).
parent(pat, brian).
female(pat).
female(jenny).
male(jim).
male(brian).
```

NOTE: do not include these in your solution file.

We assume that each person will have the same family name as their father, but that married women retain their original birth name.

Write a predicate `same_name(Person1,Person2)` that succeeds if it can be deduced from the facts in the database that `Person1` and `Person2` will have the same family name. (It is ok if your code returns `true` multiple times). For example:

```
?- same_name(pat, brian).
false.

?- same_name(jenny, jim).
true
```

Note that your `same_name` predicate will be tested with different facts to those

3. Write a predicate `log_table(NumberList, ResultList)` that binds `ResultList` to the list of pairs consisting of a number and its log, for each number in `NumberList`. For example:

```
?- log_table([1,3.7,5], Result).
Result = [[1, 0.0], [3.7, 1.308332819650179], [5, 1.6094379124341003]].
```

Note that the Prolog built-in function `log` computes the natural logarithm, and that it needs to be evaluated using `is` to actually compute the log:

```
?- X is log(3.7).
X = 1.308332819650179.

?- X = log(3.7).
X = log(3.7).
```

4. Any list of integers can (uniquely) be broken into "parity runs" where each run is a (maximal) sequence of consecutive even or odd numbers within the original list. For example, the list

```
List = [8,0,4,3,7,2,-1,9,9]
```

can be broken into `[8, 0, 4]`, `[3, 7]`, `[2]` and `[-1, 9, 9]`

Write a predicate `paruns(List, RunList)` that converts a list of numbers into the corresponding list of parity runs. For example:

```
?- paruns([8,0,4,3,7,2,-1,9,9], RunList).
RunList = [[8, 0, 4], [3, 7], [2], [-1, 9, 9]]
```

Note: you can find out how to test if a number is even or odd from the Prolog Dictionary

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
```
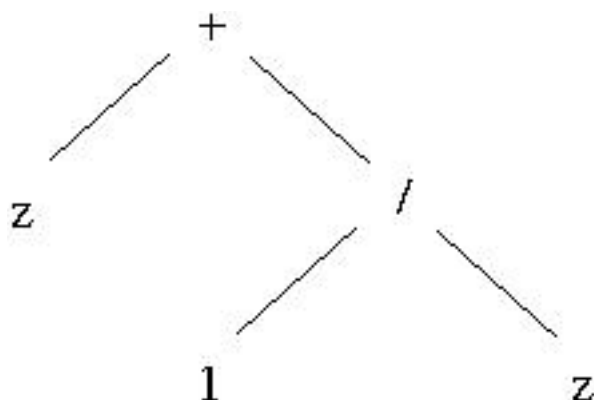

Illustration of the tree used in the example above.

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.

COMP9814 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 cs9814 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
% 9814 classrun -check prolog
```

The submission deadline is Friday 27 January, 11:59 pm.
10% 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 t.zrimec@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!