COMP3411/9414 Artificial Intelligence

Session 1, 2019

Assignment 1 - Prolog Programming

Due: Sunday 24 March, 11:59pm
Marks: 12% of final assessment for COMP3411/9414 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_list`, 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 think of additional test data in order to thoroughly test your code. Testing, and thinking of appropriate test cases, is part of the total programming task.

Note: You are **not** required to include test cases with your submission. If they are in your code, you should either comment them out or remove them prior to submitting.

Style and Exclusions

For this assignment, you are NOT allowed to use disjunctives (;) or cuts (!) and you are NOT allowed to use the built-in predicates append(), findall() or maplist().

Specifically, the characters ';', '!' and the strings "append", "findall", "maplist" must NOT occur anywhere in your file (not even in the comments). Otherwise, marks will be deducted.

It is important to use <u>exactly</u> 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
```

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 using the same format as `family.pl`

```
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 in `family.pl`

---

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

```
?- sqrt_list([0,2,289], Result).
Result = [[0, 0.0], [2, 1.4142135623730951], [289, 17.0]].
```

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

```
?- X is sqrt(5).
X = 2.23606797749979.
```

```
?- X = sqrt(5).
X = sqrt(5).
```

---

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

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

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

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

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

---

5. In this question we consider binary trees which are represented as either `empty` or `tree(L, Num, R)`, where `L` and `R` are the left and right subtrees and `Num` is a number. A binary tree of numbers is called a *heap* (or, it is said to satisfy the *heap property*) if, for every non-leaf node in the tree, the number stored at that node is less than or equal to the number stored at each of its children. For example, the following tree satisfies the heap property, because $3 \leq 5$, $5 \leq 8$ and $5 \leq 7$.

```
tree(empty, 3, tree(tree(empty, 8, empty), 5, tree(empty, 7, empty)))
```

On the other hand, the following tree does not satisfy the heap property, because 6 is not less than or equal to 5.

```
tree(tree(tree(empty, 4, empty),
    3, tree(empty, 5, empty)), 6, tree(tree(empty, 9, empty), 7, empty))
```

Write a predicate `is_heap(Tree)` which returns `true` if `Tree` satisfies the heap property, and `false` otherwise. For example:

```
?- is_heap(tree(tree(tree(empty, 4, empty),
    3, tree(empty, 5, empty)), 6, tree(tree(empty, 9, empty), 7, empty))).
false.

?- is_heap(tree(empty, 3, tree(tree(empty, 8, empty), 5, tree(empty, 7, empty)))).
true
```

Testing

This assignment will be tested primarily by an automated testing system.

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.

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 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.

When you submit, your code will be run through a small number of preliminary tests, mainly to check there are no syntax errors, your predicates are correctly named and in the right format, etc.

You can submit as many times as you like - later submissions will overwrite earlier ones.

Don't wait until the last day to submit! Submitting early and often (even a partial solution) may save you time and effort by helping you to identify potential problems early.

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 Sunday 24 March, 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!