The University of Melbourne
School of Computing and Information Systems

Declarative Programming
COMP90048

Semester 1, 2019

Project Specification

Due Thursday, 2 May 2019, 5:00PM
Worth 5%

The objective of this project is to practice your Prolog programming skills. You will write a few fairly simple Prolog predicates.

The Assignment

You will implement the following Prolog predicates.

1. `correspond(E1, L1, E2, L2)`
   this holds when in one place where list `L1` has the value `E1`, `L2` has `E2`. This must work in any mode in which `L1` and `L2` are proper lists (that is, either `[]` or a list whose tail is a proper list). For example:

   `correspond(e,[h,e,l,l,o],X,[1,2,3,4,5])` should have only the solution `X = 2`.

   `correspond(l,[h,e,l,l,o],X,[1,2,3,4,5])` should have solutions `X = 3` and `X = 4`.

   `correspond(X,[h,e,l,l,o],4,[1,2,3,4,5])` should have only the solutions `X = l`.

   `correspond(X,[h,e,l,l,o],Y,[1,2,3])` should have solutions: `X = h, Y = 1` and `X = e, Y = 2` and `X = l, Y = 3`

   `correspond(o,[g,o,o,d,b,y,e],l,[h,e,l,l,o]` should succeed.

   `correspond(y,[g,o,o,d,b,y,e],X,[h,e,l,l,o]` should fail.

2. `interleave(Ls, L)`
   this holds when `Ls` is a list of lists, and `L` is a list of all the elements of all the lists in `Ls`, interleaved. That is, the first element of `L` is the first element of the first list in `Ls`, the second element of `L` is the first element of the second list in `Ls`, and so on for all the lists in `Ls`. After this, the next element of `L` is the second element of the first list in `Ls`, and so on, until two elements have been taken from all the lists in `Ls`. After this, come the third elements of all the lists in `Ls`, and so on, until all the elements of all the lists in `Ls` are included in `L`. All the lists in `Ls` must be the same length. For example:

   `interleave([[a,b],[c,d]],X)` should have only the solution `X=[a,c,b,d]`.

   `interleave([X,Y,Z],[a,b,c,d,e,f])` should have only the solution `X=[a,d], Y=[b,e], Z=[c,f]`.

   `interleave([L],[1,2,3])` should have only the solution `L=[1,2,3]`.

`interleave([[3,1,2],[1,5],[4,9,6]],X)` should fail.

3. `partial_eval(Expr0, Var, Val, Expr)`
   this holds when `Expr` is the arithmetic expression `Expr0` with atom `Var` replaced by number `Val`, and any wholly numeric subexpressions fully evaluated to numbers. In this context an "arithmetic expression" is either:

   - a number; or
   - an atom (standing for an arithmetic variable); or
   - a term of the form x`+`y, where x and y are arithmetic expressions; or
   - a term of the form x`-`y, where x and y are arithmetic expressions; or
   - a term of the form x`*`y, where x and y are arithmetic expressions; or
   - a term of the form x`/`y, where x and y are arithmetic expressions; or
   - a term of the form x`//`y, where x and y are arithmetic expressions.

   Note that we are using Prolog atoms, rather than Prolog variables, for arithmetic variables, thus you will need to traverse `Expr0` replacing occurrences of `Var` with `Val`, and then evaluate any subexpressions containing no atoms. You need not, and should not, take advantage of algebraic equivalences like arith`x*0` $= 0$ and arith`x-x` $= 0$; if `x` is not substituted with a value, both of these cases should be left as is in the output. Similarly, you should not exploit algebraic properties such as associativity, commutativity or distributivity, so expressions such as `6*x*7` should also be left as is. For example:

   `partial_eval(6*7, x, 2, E)` should have only the solution `E=42`.

   `partial_eval(6*(3+x*x), x, 2, E)` should have only the solution `E=42`.

   `partial_eval(x*(3+y*y), y, 2, E)` should have only the solution `E=x*7`.

   `partial_eval((x*0+6)*(x-x+3+y*y), y, 2, E)` should have only the solution `E=(x*0+6)*(x-x+3+4)`.

   This predicate need only work when `Expr0`, `Var`, and `Val` are ground, and it should produce only one solution for `Expr`.

   Hint: Prolog typically does not use a special constructor to indicate a number or an atom, as one would with an algebraic type system. In Prolog, you can use the built-in predicates `number(X)` and `atom(X)` to test if the term `X` is a number or atom, respectively.

Note that the order in which solutions are found does not matter, but all listed answers must be found, and no extra solutions are permitted.

You must call your source file `assignment2.pl`.

Assessment

Your project will be assessed 100% on correctness. For this assignment, code quality will not be considered. However, for your own sanity, I do recommend commenting your code and programming it carefully, paying due attention to programming technique.

Note that timeouts will be imposed on all tests. Test cases will be rather small, so the timeouts should only affect you if you create an infinite recursion (infinite loop) or infinite backtracking loop.

Submission

You **must** submit your project from either of the unix servers `dimefox.eng.unimelb.edu.au` or `nutmeg.eng.unimelb.edu.au`. Make sure the version of your program source file you wish to submit is on this host, then `cd` to the directory holding your source code and issue the command:

```
submit COMP90048 assignment2 assignment2.pl
```

Important: you must wait a minute or two (or more if the servers are busy) after submitting, and then issue the command

```
verify COMP90048 assignment2 | less
```

This will show you the test results from your submission, as well as the file(s) you submitted. If the test results show any problems, correct them and submit again. You may submit as often as you like; only your final submission will be assessed.

If you wish to (re-)submit after the project deadline, you may do so by adding "`.late`" to the end of the project name (*i.e.,* `assignment2.late`) in the `submit` and `verify` commands. But note that a penalty, described below, will apply to late submissions, so you should weigh the points you will lose for a late submission against the points you expect to gain by revising your program and submitting again.

It is your responsibility to verify your submission.

Your submission will be tested on one of the servers `dimefox2.eng.unimelb.edu.au` or `nutmeg2.eng.unimelb.edu.au` (note the "2"). These servers run SWI Prolog version 7.6.4, which is probably older than the version you will develop on. You are advised to test your program on one of these servers before submitting; in the unlikely case that your program uses some Prolog features or libraries not supported by SWI 7.6.4, it will be much easier to discover this.

Note that these hosts are only available through the university's network. If you wish to use these machines from off campus, you will need to use the university's Virtual Private Network. The LMS Resources list gives instructions.

Windows users should see the LMS Resources list for instructions for downloading the (free) MobaXterm or Putty and Winscp programs to allow you to use and copy files to the department servers from windows computers. Mac OS X and Linux users can use the `ssh`, `scp`, and `sftp` programs that come with your operating system.

Late Penalties

Late submissions will incur a penalty of 0.5% of the possible value of that submission per hour late, including evening and weekend hours. This means that a perfect project that is much more than 4 days late will receive less than half the marks for the project. If you have a medical or similar compelling reason for being late, you should contact the lecturer as early as possible to ask for an extension (preferably before the due date).

Note Well:

This project is part of your final assessment, so cheating is not acceptable. Any form of material exchange between teams, whether written, electronic or any

other medium, is considered cheating, and so is the soliciting of help from electronic newsgroups. Providing undue assistance is considered as serious as receiving it, and in the case of similarities that indicate exchange of more than basic ideas, formal disciplinary action will be taken for all involved parties. If you have questions regarding these rules, please ask the lecturer.