ECE/CPSC 3520
SS II 2020
Software Design Exercise #2
Solving Problems Inspired by Google, Amazon, Microsoft, etc. Coding Interviews Using `Prolog`

Canvas submission only
Assigned 7/17/2020
Revised 7/12/2020
Due 7/27/2020 11:59PM

Contents

1 Preface

SDE2 concerns the same problems as SDE1 but with a different programming paradigm and implementation language. The overall objective of SDE2 is to implement 4 (sets of) predicates corresponding to 4 different list-based problems. Three of these problems were inspired by actual coding tests given during interviews for positions with companies such as Google, Amazon, Microsoft and Apple. This is to be done using a purely declarative programming paradigm and `SWI-Prolog`. Note that a Canvas video lecture accompanies this assignment.
Notes:

1. Some of the problems which inspired this assignment refer to arrays or strings. **The common (and required) data structure in this assignment is the `Prolog` list.**
2. **The videos provided are only to characterize the problem.** The video solutions presented are imperative, i.e., neither functional (`ocaml` in SDE1) or declarative (Prolog in SDE2). Typically pointers and/or imperative (java, c) solutions are shown. Thus, the video solution probably has little to do with your declarative (SDE2) solution.

The problems are:

first duplicate in a list problem :

`https://www.youtube.com/watch?v=XSdr_O-XVRQ`

Note, like in SDE1, we are implementing a modified solution to the 'closest duplicate' problem cited in the video.

first non-repeating in a list problem :

`https://youtu.be/5co5Gvp_-S0`

the sum of 2 problem :

`https://www.youtube.com/watch?v=BoHO04xVeU0`

`https://www.youtube.com/watch?v=sfuZzBLPcx4`

CYK parsing algorithm-inspired list decomposition problem :

> Given n, return a list of 2-element lists, each indicating how a string of length n could be formed from 2 strings. [[1,n-1], [2,n-2],...,[n-1,1]] is the returned `Prolog` list.

This should actually be some fun and really good experience with coding interview tests. Each of the problems is described separately, with examples.

2 The 4 Problems and Required Top-Level `Prolog` Predicates for Each

2.1 Specifying Prototypes

Recall predicate arity is indicated with slash notation, as used in the prototype specifications below.

```
/* Recall the argument designation from SWI-Prolog, where:

+Argument means the argument is bound before invoking the predicate

-Argument means the argument is bound to a value, if possible,
in the process of finding a solution

?Argument means either (works both ways)
*/
```

Notes:

1. The arity and argument specification is only used to convey the prototype, **this notation is NOT used in invoking the predicates**. This is shown in the examples; and
2. Only the top-level predicates are specified. It is reasonable to expect you may need to design, implement and test other (supporting or auxiliary) predicates for each. These will not be specified or tested directly.

2.2 First Duplicate in a List (`first_duplicate/2`)

Pay special attention to the naming and argument(s) of each required `Prolog` predicate.
You can see the problem formulation at:

`https://www.youtube.com/watch?v=XSdr_O-XVRQ`

Note: The interpretation of the `first_duplicate` predicate here is the same as the one (intended) for function `first_duplicate` in SDE1:

first_duplicate suceeds and binds argument Dup to the first element in Alist with a duplicate (in the remainder of the Alist). The video I referenced really describes a different algorithm ('closest duplicate'). Like SDE1, the problem solution is much easier with this interpretation..

```
Prototype: first_duplicate(+Alist,-Dup)

if succeeds, binds Dup to first duplicate in Alist,
fails otherwise (no duplicates)

Sample Use:

?- first_duplicate([1,2,3,4,5,6,7,4,5,8,9],W).
W = 4.

?- first_duplicate([1,2,3,4,5,6,7,4,5,2,9],W).
W = 2.

?- first_duplicate([1,2,3,4,5,6,7,8,9,10],W).
false.
```

2.3 First Non-Repeating Element in a List (`first_nonrepeating/2`)

You can see the problem formulation at:

`https://youtu.be/5co5Gvp_-S0`

```
Prototype: first_nonrepeating(+Alist,-Which). */
"Determine the first non-repeated element (Which) in the *entire* list" (Alist).
fails if no repeated elements in Alist.

Samples:
```

```
?- first_nonrepeating([1,2,3,2,7,5,6,1,3],Who).
Who = 7.

?- first_nonrepeating([1,2,7,3,2,7,5,6,1,3],Who).
Who = 5.

?- first_nonrepeating([1,5,2,7,3,2,7,5,6,1,3],Who).
Who = 6.

?- first_nonrepeating([1,5,2,7,3,2,7,5,6,6,1,3],Who).
false.

?- first_nonrepeating([1,5,2,7,3,2,7,5,6,6,3],Who).
Who = 1.

?- first_nonrepeating([1,5,7,3,2,7,5,6,6,3],Who).
Who = 1.
```

2.4 The Sum of 2 Problem (`sum_of_two/3`)

You can see the problem formulation at:
    `https://www.youtube.com/watch?v=BoHO04xVeU0`
   `https://www.youtube.com/watch?v=sfuZzBLPcx4`
Two arrays contain numbers able to produce a given sum. A sample case from the video:

```
If a=[1;2;3], b=[10;20,30,40] and v=42, then sum_of_two(a,b,v)=true
since 40+2)=42=v.
```

```
Prototype: sum_of_two(+A,+B,+V)

Samples:

?- sum_of_two([4,2],[79,30],32).
true.

?- sum_of_two([4,2],[79,30],31).
false.

?- sum_of_two([1,2,3],[10,20,30,40],42).
true.

?- sum_of_two([10,20,30,40],[1,2,3],42).
```

```
true.

?- sum_of_two([1,2,3,4,5],[10,20,30,40],26).
false.

?- sum_of_two([1,2,3,4,5],[10,20,30,40],27).
false.
```

2.5 CYK Parsing Algorithm-Inspired Problem (cyk_sublists/2)

Please note we are NOT trying to implement the CYK algorithm in `ocaml`. However, in forming the CYK parse table (Book, Chapter 4, Section 4.4.6) it is necessary to consider how many ways a string (here a list) of length `n` may be comprised by concatenation of two non-empty sublists.
Given `n`, `cyk_sublists` returns a list of two-element lists (Recall tuples were used in SDE1.) Each 2-element sublist consists of the lengths of the 2 component strings which are concatenated to form the `n` element list.

```
Prototype: cyk_sublists(+N,-Decompositions)
Given N, return a list of lists (Decompositions), where
each sublist indicates how a string of length N could be formed from 2 strings.
[(1,n-1], [2,n-2],...,[n-1,1]] is returned Prolog list

Samples:

?- ['cyk_sublists.pro'].
true.

?- cyk_sublists(5,Result).
Result = [[1, 4], [2, 3], [3, 2], [4, 1]].

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

?- cyk_sublists(3,Result).
Result = [[1, 2], [2, 1]].

?- cyk_sublists(2,Result).
Result = [[1, 1]].

?- cyk_sublists(10,Result).
Result = [[1, 9], [2, 8], [3, 7], [4, 6], [5, 5], [6, 4], [7, 3], [8|...], [...|...]].
```

3 Resources

It would be foolish to attempt this SDE without carefully exploring:

1. The text, especially the many examples in Chapter 3;
2. `http://www.swi-prolog.org/`;
3. The `SWI-Prolog` manual;
4. The course Prolog lectures and corresponding videos on Canvas; and
5. The experience of SDE1.

4 How We Will Build and Evaluate Your Prolog Solution

First, we'll consult your Prolog source file to identify any warnings or errors. Then, a Prolog script will be used with varying input files and test vectors to test your predicates, as shown in the examples.

BE SURE YOUR PREDICATES CONFORM TO THE SPECIFIED PROTOTYPES!

We will not rewrite or revise your submission to facilitate grading. The grade is based upon a correctly working solution which is submitted by the deadline.

5 Pragmatics and Constraints

Use this document as a checklist to be sure you have responded to all parts of the assignment, and have included the required files (readme.txt, sde2.pro and sde2.log).

- The simulation uses `only Prolog code` you wrote (excluding the given predicates).
- **No use of Prolog's 'if..then' construct** (written as `->/2`):

```
:Condition -> :Action  construct

(A -> B ; C). /* this is if-then else */
```

- No use of `assert` or `retract` predicates

6 Format of the Electronic Submission

The final **zipped** archive is to be named `<yourname>`**-sde2.zip**, where `<yourname>` is your (CU) assigned user name. You will upload this to the Canvas assignment prior to the deadline.

The minimal contents of this archive are as follows:

1. A `readme.txt` file listing the contents of the archive and a brief description of each file. Include 'the pledge' here. Here's the pledge:

   Pledge:
   On my honor I have neither given nor received aid on this exam.

   This means, among other things, that the code you submit is **your** code.

2. The `Prolog` source file for your implementation (named `sde2.pro`). Note this file must include the predicates defined in Section 2, as well as any additional predicates you design and implement. This is the file our Prolog scripts will consult.

3. An ASCII log file showing 2 sample uses of each of the predicates required in Section 2. Name this log file `sde2.log`.

The use of `Prolog` should not generate any errors or warnings. The grader will attempt to interpret your Prolog source and check for correct behavior. Recall the grade is based upon a correctly working solution.

7 Final Remark: Deadlines Matter

This remark was included in the course syllabus and SDE1. Since multiple submissions to Canvas are allowed[1], if you have not completed all predicates, you should submit a freestanding archive of your current success before the deadline. This will allow the possibility of partial credit. **Do not attach any late submissions to email and send them to either me or the graders.**

[1]But we will only download and grade the latest (or most recent) one, and it must be submitted by the deadline.