ECE/CPSC 3520
Spring 2018
Software Design Exercise #1

Canvas submission only
Assigned 2/6/2018
(Previewed 2/1/2018)
Due 3/6/2018

Contents

1 Preface

The overall objectives of SDE1 are:

1. To gain experience in declarative logic as a programming paradigm; and
2. To gain experience in the design, implementation and testing of Prolog representations.

Your grade is based upon a correctly working implementation. **Get started now.**[1] This is an individual (not group) effort.

2 Resources

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

- The book, esp. Chapter 3 (Declarative Programming and Prolog);
- The SWI-Prolog Manual (embedded in SWI-Prolog); and
- Related course lectures.

2.1 Objectives

The objective of SDE 1 is to design, implement and test aspects of, and related to, the CYK parsing algorithm in `Prolog`. We will not implement all of the CYK table formation in Prolog this semester.

This document specifies a number of predicates which must be developed as part of the effort. **This SDE MUST be done using only `SWI-Prolog` on a linux platform**. It probably makes a lot of sense to review the hand formation of the CYK table before beginning the Prolog implementation.

2.2 Standard Remarks

Please note:

1. This assignment assesses *your* effort (not mine). I will not debug or design your code, nor will I install software for you. You (and only you) need to do these things.
2. It is never too early to get started on this effort.
3. As noted, we may occasionally discuss this in class.

[1] At least install SWI-Prolog.

3 Data Structures and Representation in `Prolog`

One of the most common questions which arise at this point is:

> *'How do I get the productions, the string to be parsed and table 'into' Prolog?'*

The following should help.

3.1 Productions and Input String

Notice Prolog strings (" ") are used for representation of the terminals and nonterminals.

```
/* named production *set*  as arity-2 predicate: productions/2
Prototype: productions(+Name,-Data).   */

productions(book, [["S","AB"],["S","BB"],["A","CC"],["A","AB"],["A","a"],
["B","BB"],["B","CA"],["B","b"],["C","BA"],["C","AA"],["C","b"]]).

/* example uses:

?- listing(productions).
productions(book, [["S", "AB"], ["S", "BB"], ["A", "CC"], ["A", "AB"],
["A", "a"], ["B", "BB"], ["B", "CA"], ["B", "b"],
["C", "BA"], ["C", "AA"], ["C", "b"]]).

true.

?- productions(book,What).
What = [["S", "AB"], ["S", "BB"], ["A", "CC"], ["A", "AB"], ["A", "a"],
["B", "BB"], ["B", "CA"], ["B"|...], [...|...]|...].

?- productions(book,What), length(What,L).
What = [["S", "AB"], ["S", "BB"], ["A", "CC"], ["A", "AB"], ["A", "a"],
["B", "BB"], ["B", "CA"], ["B"|...], [...|...]|...],
L = 11.

*/

/* string to parse: astring/2
Prototype: astring(+Name, -Data)
*/

astring(bookstring,["a", "a", "b", "b"]).

/* example uses:

?- astring(Who,What).
Who = bookstring,
What = ["a", "a", "b", "b"].

*/
```

3.2 CYK Parse Table List Structure

```
[<row_1>, <row_2>,<row_3>,...<row_n>]

where

<row_i> = [<cell_1>, <cell_2>, <cell_3>, ...<cell_j>]

and

|<row_1>| = n, |<row_2>| = n-1, ..., |<row_n>| = 1

where

cell_i = [<nonterminal_symbols>]
```

3.3 CYK table in Prolog

```
/* CYK table in Prolog
table/2
Prototype: table(+Name,-Data)
*/

/* a simple sample table */

table(sample_table4,[
[["11"],["21"],["31"],["41"]],
[["12"],["22"],["32"]],
[["13"],["23"]],
[["14"]]
]).

/* another table Predicate Example (formatted for ease of viewing) */

table(book_result, [
[["A"], ["A"], ["B", "C"], ["B", "C"]],
[["C"], ["S", "A"], ["S", "B", "A"]],
[["C", "A"], ["C", "S", "A"]],
[["C", "B", "S", "A"]]
]).

/* sample:

?- table(book_result,What).
What = [[["A"], ["A"], ["B", "C"], ["B", "C"]], [["C"], ["S", "A"],
["S", "B", "A"]], [["C", "A"], ["C", "S", "A"]], [["C", "B", "S", "A"]]].
*/
```

4 Prolog Predicates to be Designed, Implemented and Tested

You will design, implement, test and deliver the 8 predicates described below which involve aspects of CYK parsing in Prolog. Note:

1. Recall the '/n' in the predicate representation indicates the arity of the predicate is n. It is not part of the name but rather the Prolog convention for showing predicate name and arity.
2. Recall the +,- or ? symbol indicates the role of the argument.
3. Pay special attention to the predicate naming and argument specifications. Sine your submission will be graded by a Prolog script, if you deviate from this your tests will probably fail. **The graders will not fix or change anything in your submission.**
4. Pay attention to the file naming conventions specified.
5. The implementation of all predicates is to be included in a single file named `sde1.pro`.
6. Assume all productions are given in CNF (in the Prolog representation shown below.) No conversion or testing is necessary.
7. You may design and use other predicates to support any of these predicates, but they are not (directly) graded. They must also be included in your SDE1 Prolog file.

4.1 get_table_values_cell/3

For table 'access' in Prolog. Note cell contents are displayed as a list. Note some of the results are wordwrapped to fit on the page.

```
Prototype: get_table_values_cell([+I,+J],+Table,-ContentsL)

list of [i,j] used as first argument
for table indices i: element, j: length (both >=1)
```

Examples.

```
/* data to use */

table(sample_table4,[
[["11"],["21"],["31"],["41"]],
[["12"],["22"],["32"]],
[["13"],["23"]],
[["14"]]
]).
?- table(sample_table4,Data).
Data = [[["11"], ["21"], ["31"], ["41"]], [["12"],
```

```
["22"], ["32"]], [["13"], ["23"]], [["14"]]].

?- table(sample_table4,Data),get_table_values_cell([3,2],Data,CL).
Data = [[["11"], ["21"], ["31"], ["41"]], [["12"], ["22"], ["32"]],
[["13"], ["23"]], [["14"]]],
CL = ["32"].

?- table(sample_table4,Data),get_table_values_cell([1,4],Data,CL).
Data = [[["11"], ["21"], ["31"], ["41"]], [["12"], ["22"], ["32"]],
[["13"], ["23"]], [["14"]]],
CL = ["14"].

?- table(book_result,Data),get_table_values_cell([3,1],Data,CL).
Data = [[["A"], ["A"], ["B", "C"], ["B", "C"]], [["C"], ["S", "A"],
["S", "B", "A"]], [["C", "A"], ["C", "S", "A"]], [["C", "B", "S", "A"]]],
CL = ["B", "C"].

?- table(book_result,Data),get_table_values_cell([1,4],Data,CL).
Data = [[["A"], ["A"], ["B", "C"], ["B", "C"]], [["C"], ["S", "A"],
["S", "B", "A"]], [["C", "A"], ["C", "S", "A"]], [["C", "B", "S", "A"]]],
CL = ["C", "B", "S", "A"].

table(book_result, [
[["A"], ["A"], ["B", "C"], ["B", "C"]],
[["C"], ["S", "A"], ["S", "B", "A"]],
[["C", "A"], ["C", "S", "A"]],
[["C", "B", "S", "A"]]
]).
```

4.2 decompositions/2

```
/* decompositions/2
Prototype: decompositions(+N,-List_of_decomposition_sublists)

N is string length. This predicate is used for
forming list of decomposition sublists of form [j,k]
where [j,k] means a list of length j followed by a list of length k
i.e., the 'j+k' decomposition in the CYK table formation.
Note: This is not cell j,k in the table
*/
```

Examples.

```
/* samples:

?- decompositions(4,W).
W = [[3, 1], [2, 2], [1, 3]].

?- decompositions(5,W).
W = [[4, 1], [3, 2], [2, 3], [1, 4]].

?- decompositions(3,W).
W = [[2, 1], [1, 2]].

?- decompositions(2,W).
```

```
W = [[1, 1]].
*/
```

4.3 Cell Products

Recall in the CYK table formation we populate cells with the names of nonterminals capable of forming the string in two parts. We consider row 1 formation (j=1) a special case. To help you, we consider two cases.

4.3.1 one_product/3

This involves a string and a cell:

```
/* first, product of one nonterminal (not a list)
and another cell (list) contents */

one_product(+Nonterminal,+Cell,-Product)
```

Examples.

```
?- one_product([],["B"],What).
What = [].

?- one_product("A",["B"],What).
What = ["AB"].      /* note this is NOT ["A","B"] */

?- one_product("A",["D","F"],What).
What = ["AD", "AF"].

?- one_product("A",[],What).
What = [].
```

4.3.2 cell_products/3

Now consider the 'outer product' of 2 cells. This is used extensively in the CYK algorithm. Either cell could be empty.

```
cell_products(+Cell1,+Cell2,-Product)
```

Examples.

```
?- cell_products([],[],W).
W = [].

?- cell_products([],["A"],W).
W = [].

?- cell_products(["A"],[],W).
W = [].

?- cell_products(["A"],["C"],What).
What = ["AC"].

?- cell_products(["A","B"],["C","D"],What).
```

```
What = ["AC", "AD", "BC", "BD"].

?- cell_products(["A","B","C","D"],["F","G"],What).
What = ["AF", "AG", "BF", "BG", "CF", "CG", "DF", "DG"].
```

4.4 form_row1_cell/3

This predicate is used for forming a single cell in the first row of the CYK table. This cell corresponds to string element x_i.

```
Prototype: form_row1_cell(+StringElement,+ProductionsList,-Row1Cell)

   We treat forming 1st row as special case --
   this corresponds to only one input string element per cell
   so there are no RHS nonterminals in productions to consider.
   This is defacto scanning.
   Returns contents of a single row1 cell upon success
   as the third argument binding.
```

Example.

```
?- productions(book,BookProds),form_row1_cell("b",BookProds,Cell).
BookProds = [["S", "AB"], ["S", "BB"], ["A", "CC"], ["A", "AB"],
["A", "a"], ["B", "BB"], ["B", "CA"], ["B"|...], [...|...]|...],
Cell = ["B", "C"].

?- productions(book,BookProds),form_row1_cell("a",BookProds,Cell).
BookProds = [["S", "AB"], ["S", "BB"], ["A", "CC"], ["A", "AB"],
["A", "a"], ["B", "BB"], ["B", "CA"], ["B"|...], [...|...]|...],
Cell = ["A"].

?- productions(book,BookProds),form_row1_cell("c",BookProds,Cell).
BookProds = [["S", "AB"], ["S", "BB"], ["A", "CC"], ["A", "AB"],
["A", "a"], ["B", "BB"], ["B", "CA"], ["B"|...], [...|...]|...],
Cell = [].
```

4.5 Equivalence of 2 CYK Parse Tables

Note the contents of any cell in a CYK table are a set of nonterminal names. However, in using lists, an order will result. Thus, 2 different lists may represent the same cell. In the next 2 predicates, we check for equivalence in 3 parts.

4.5.1 equivalent/2

This tests for equivalence of 2 cells.

```
Prototype: equivalent(+A,+B)
succeeds if contents of the cells (lists) are identical
```

Examples.

```
?- equivalent(["B", "C"],["C", "B"]).
true.

?- equivalent([],[]).
true.

?- equivalent(["S","B","A"],["B","S","A"]).
true.

?- equivalent(["S","B","A"],["B","S","A","C"]).
false.

?- equivalent([],["B","S","A","C"]).
false.

?- equivalent(["B","S","C","A"],["B","D","S","C","A"]).
false.
```

4.5.2 row_equivalent/2

Here we apply the constraint two rows (each corresponding to the same j, or string length).

```
Prototype: row_equivalent(+RowA,+RowB)
Succeeds if rows are equivalent (i.e., all corresponding cells are equivalent).
```

Examples.

```
?- row_equivalent([[],[],[]],[[],[],[]]).
true.

?- row_equivalent([[],["A"],[]],[[],[],["A"]]).
false.

?- row_equivalent([["A"], ["A"], ["B", "C"], ["B", "C"]],
[["A"], ["A"], ["B", "C"], ["B", "C"]]).
true.

?- row_equivalent([["A"], ["A"], ["B", "C"], ["B", "C"]],
[["A"], ["A"], ["C", "B"], ["C", "B"]]).
true.

?- row_equivalent([["A"], ["A"], ["B", "C"], ["B", "C"]],
[[], ["A"], ["C", "B"], ["C", "B"]]).
false.
```

4.5.3 table_equivalent/2

```
Prototype: table_equivalent(+TableA,+TableB)
Succeeds if Tables are equivalent (i.e., all corresponding rows are equivalent).
```

Examples.

```
/* data */

table(book,[[["A"], ["A"], ["B", "C"], ["B", "C"]],
 [["C"], ["S", "A"], ["S", "B", "A"]],
 [["C", "A"], ["C", "S", "A"]],
 [["C", "B", "S", "A"]]]).

table(book3,[[["A"], ["A"], ["B", "C"], ["B", "C"]],
 [["C"], ["S", "A"], ["S", "B", "A"]],
 [["C", "A"], ["C", "S", "A"]],
 [["A", "S", "B", "C"]]]).

table(bookbad1,[[["A"], ["A"], ["B", "C"], ["B", "C"]],
 [["C"], ["S", "A"], ["S", "B", "A"]],
 [["C", "A"], ["C", "S", "A"]],
 [["C", "B", "S", "G"]]]).

table(bookbad2,[[["A"], ["A"], ["B", "C"], ["B", "C"]],
 [["C"], ["S", "A"], ["S", "B", "A"]],
 [["C", "A"], ["C", "A"]],
 [["C", "B", "S", "A"]]]).

?- table(book,T1),table(book3,T2),table_equivalent(T1,T2).
T1 = [[["A"], ["A"], ["B", "C"], ["B", "C"]], [["C"], ["S", "A"],
["S", "B", "A"]], [["C", "A"], ["C", "S", "A"]], [["C", "B", "S", "A"]]],
T2 = [[["A"], ["A"], ["B", "C"], ["B", "C"]], [["C"], ["S", "A"],
["S", "B", "A"]], [["C", "A"], ["C", "S", "A"]], [["A", "S", "B", "C"]]].

?- table(book3,T1),table(bookbad1,T2),table_equivalent(T1,T2).
false.

?- table(book3,T1),table(bookbad2,T2),table_equivalent(T1,T2).
false.
```

5 Prolog Use Limitations and Constructs Not Allowed

1. The entire solution must be in SWI-Prolog (Version 7.2 or newer.)

2. No use of the `if..then` construct (Don't even bother looking for it). It has the syntax:

   ```
   condition -> then_clause ; else_clause
   ```

   and is useful for people who want an if-then capability, but don't understand the goal-satisfaction mechanism in Prolog.

3. No use of predicates `assert` or `retract`.

   If you are in doubt about any allowable predicates, ask and I'll provide a 'private-letter ruling'.

The objective is to obtain proficiency in declarative programming and Prolog, not to try to find built-in predicates which simplify or trivialize the effort, or to implement an imperative solution in a declarative programming paradigm.

6 Pragmatics

Use this document as a checklist to be sure you have responded to all parts of the SDE, and have included the required files (*.pro and *.log). Furthermore:

- The simulation uses only (SWI-Prolog) code you wrote.
- The final, single zipped archive which must be submitted (to Canvas) by the deadline must be named **<yourname>-sde1.zip**, where <yourname> is your (CU) assigned user name.

  The contents of this archive are:

  1. A `readme.txt` file listing the contents of the archive and a very 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
     SIGN _____

  2. Your single SWI-Prolog source file `sde1.pro` containing the required predicates.
  3. A log file named `sde1.log` showing 3 uses of each required predicate. **Uses test cases other than those shown herein.**

We will attempt to consult your Prolog source file and then query Prolog with relevant goals. Most of the grade will be based upon this evaluation.