Prolog Exercises from LPN

Section 2.3 - exercises

Exercise 2.2 We are working with the following knowledge base:

```
house_elf(dobby).
witch(hermione).
witch('McGonagall').
witch(rita_skeeter).
magic(X):-  house_elf(X).
magic(X):-  wizard(X).
magic(X):-  witch(X).
```

Which of the following queries are satisfied? Where relevant, give all the variable instantiations that lead to success.

1. ?- magic(house_elf).
2. ?- wizard(harry).
3. ?- magic(wizard).
4. ?- magic('McGonagall').
5. ?- magic(Hermione).

Draw the search tree for the query magic(Hermione) .

Exercise 2.3 Here is a tiny lexicon (that is, information about individual words) and a mini grammar consisting of one syntactic rule (which defines a sentence to be an entity consisting of five words in the following order: a determiner, a noun, a verb, a determiner, a noun).

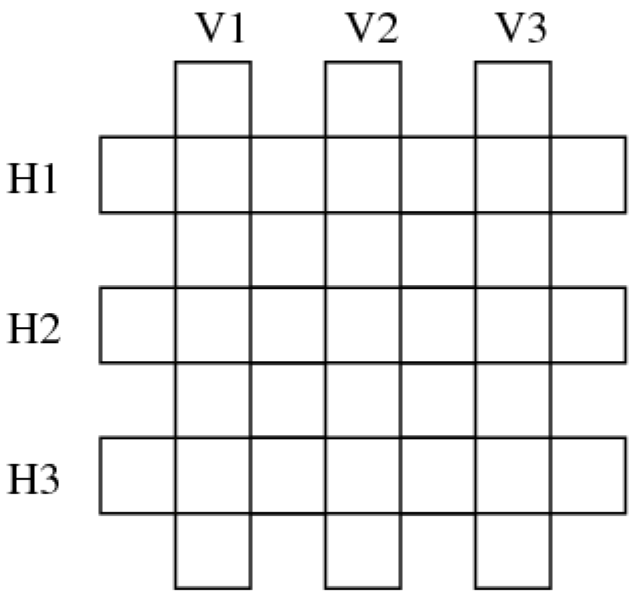
```
word(determiner,a).
word(determiner,every).
word(noun,criminal).
word(noun,'big  kahuna  burger').
word(verb,eats).
word(verb,likes).

sentence(Word1,Word2,Word3,Word4,Word5):-
      word(determiner,Word1),
      word(noun,Word2),
      word(verb,Word3),
      word(determiner,Word4),
      word(noun,Word5).
```

What query do you have to pose in order to find out which sentences the grammar can generate? List all sentences that this grammar can generate in the order that Prolog will generate them in.

Exercise 2.4 Here are six Italian words:

astante , astoria , baratto , cobalto , pistola , statale .

They are to be arranged, crossword puzzle fashion, in the following grid:



The following knowledge base represents a lexicon containing these words:

```
word(astante,  a,s,t,a,n,t,e).
word(astoria,  a,s,t,o,r,i,a).
word(baratto,  b,a,r,a,t,t,o).
word(cobalto,  c,o,b,a,l,t,o).
word(pistola,  p,i,s,t,o,l,a).
word(statale,  s,t,a,t,a,l,e).
```

Write a predicate crossword/6 that tells us how to fill in the grid. The first three arguments should be the vertical words from left to right, and the last three arguments the horizontal words from top to bottom.

Section 3.3 - exercises

Exercise 3.1 In the text, we discussed the predicate

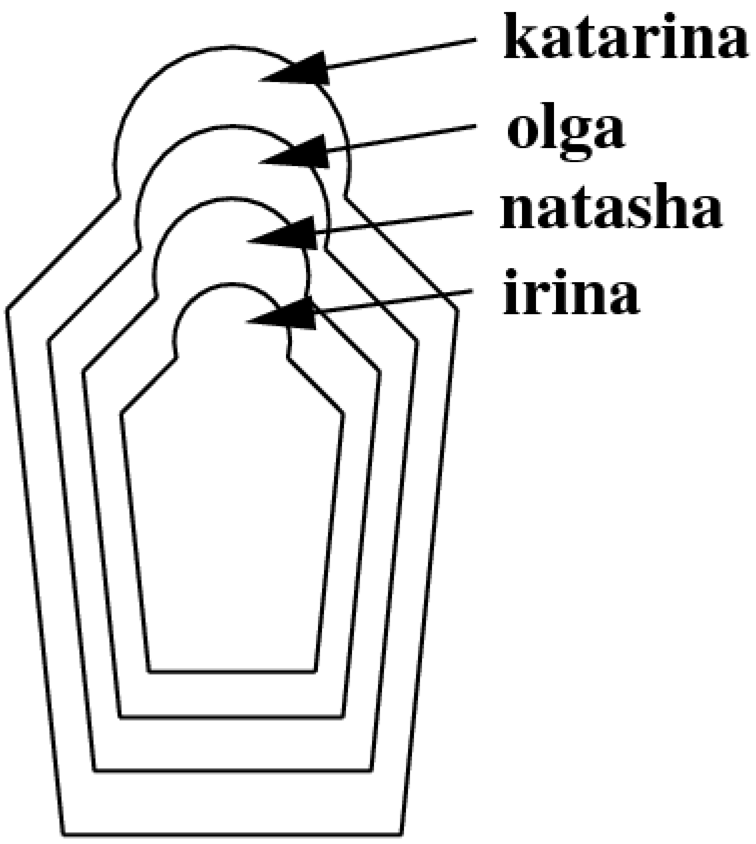
```
descend(X,Y)  :-  child(X,Y).
descend(X,Y)  :-  child(X,Z),
                  descend(Z,Y).
```

Suppose we reformulated this predicate as follows:

```
descend(X,Y)  :-  child(X,Y).
descend(X,Y)  :-  descend(X,Z),
                  descend(Z,Y).
```

Would this be problematic?

Exercise 3.2 Do you know these wooden Russian dolls (Matryoshka dolls) where the smaller ones are contained in bigger ones? Here is a schematic picture:



First, write a knowledge base using the predicate directlyIn/2 which encodes which doll is directly contained in which other doll. Then, define a recursive predicate in/2 , that tells us which doll is (directly or indirectly) contained in which other dolls. For example, the queryin(katarina,natasha) should evaluate to true, while in(olga, katarina) should fail.

Exercise 3.3 We have the following knowledge base:

```
directTrain(saarbruecken,dudweiler).
directTrain(forbach,saarbruecken).
directTrain(freyming,forbach).
directTrain(stAvold,freyming).
directTrain(fahlquemont,stAvold).
directTrain(metz,fahlquemont).
directTrain(nancy,metz).
```

That is, this knowledge base holds facts about towns it is possible to travel between by taking a direct train. But of course, we can travel further by chaining together direct train journeys. Write a recursive predicate travelFromTo/2 that tells us when we can travel by train between two towns. For example, when given the query

```
travelFromTo(nancy,saarbruecken).
```

it should reply yes.

Section 3.4 - tasks

Extend the predicate travel/3 so that it not only tells you the route to take to get from one place to another, but also how you have to travel. That is, the new program should let us know, for each stage of the voyage, whether we need to travel by car, train, or plane.

1. Imagine that the following knowledge base describes a maze. The facts determine which points are connected, that is, from which points you can get to which other points in one step. Furthermore, imagine that all paths are one-way streets, so that you can only walk them in one direction. So, you can get from point 1 to point 2, but not the other way round.

```
connected(1,2).
connected(3,4).
connected(5,6).
connected(7,8).
connected(9,10).
connected(12,13).
connected(13,14).
connected(15,16).
connected(17,18).
connected(19,20).
connected(4,1).
connected(6,3).
connected(4,7).
connected(6,11).
connected(14,9).
connected(11,15).
connected(16,12).
connected(14,17).
connected(16,19).
```

   Write a predicate path/2 that tells you from which points in the maze you can get to which other points when chaining together connections given in the above knowledge base. Can you get from point 5 to point 10? Which other point can you get to when starting at point 1? And which points can be reached from point 13?

2. We are given the following knowledge base of travel information:

```
byCar(auckland,hamilton).
byCar(hamilton,raglan).
byCar(valmont,saarbruecken).
byCar(valmont,metz).

byTrain(metz,frankfurt).
byTrain(saarbruecken,frankfurt).
byTrain(metz,paris).
byTrain(saarbruecken,paris).

byPlane(frankfurt,bangkok).
byPlane(frankfurt,singapore).
byPlane(paris,losAngeles).
byPlane(bangkok,auckland).
byPlane(singapore,auckland).
byPlane(losAngeles,auckland).
```

   Write a predicate travel/2 which determines whether it is possible to travel from one place to another by chaining together car, train, and plane journeys. For example, your program should answer yes to the query travel(valmont,raglan) .

3. So, by using travel/2 to query the above database, you can find out that it is possible to go from Valmont to Raglan. If you are planning such a voyage, that's already something useful to know, but you would probably prefer to have the precise route from Valmont to Raglan. Write a predicate travel/3 which tells you which route to take when travelling from one place to another. For example, the program should respond

```
X = go(valmont,metz,
        go(metz,paris,
           go(paris,losAngeles)))
```

to the query travel(valmont,losAngeles,X) .

4. Extend the predicate travel/3 so that it not only tells you the route to take to get from one place to another, but also how you have to travel. That is, the new program should let us know, for each stage of the voyage, whether we need to travel by car, train, or plane.

Section 4.4

Exercise 4.5 Suppose we are given a knowledge base with the following facts:

```
tran(eins,one).
tran(zwei,two).
tran(drei,three).
tran(vier,four).
tran(fuenf,five).
tran(sechs,six).
tran(sieben,seven).
tran(acht,eight).
tran(neun,nine).
```

Write a predicate listtran(G,E) which translates a list of German number words to the corresponding list of English number words. For example:

```
listtran([eins,neun,zwei],X).
```

should give:

```
X = [one,nine,two].
```

Your program should also work in the other direction. For example, if you give it the query

```
listtran(X,[one,seven,six,two]).
```

it should return:

```
X = [eins,sieben,sechs,zwei].
```

(Hint: to answer this question, first ask yourself "How do I translate the empty list of number words?". That's the base case. For non-empty lists, first translate the head of the list, then use recursion to translate the tail.)

Exercise 4.6 Write a predicate twice(In,Out) whose left argument is a list, and whose right argument is a list consisting of every element in the left list written twice. For example, the query

```
twice([a,4,buggle],X).
```

should return

```
X = [a,a,4,4,buggle,buggle]).
```

And the query

```
twice([1,2,1,1],X).
```

should return

```
X  =  [1,1,2,2,1,1,1,1].
```

(Hint: to answer this question, first ask yourself “What should happen when the first argument is the empty list?”. That’s the base case. For non-empty lists, think about what you should do with the head, and use recursion to handle the tail.)

Exercise 4.7 Draw the search trees for the following three queries:

```
?-  member(a,[c,b,a,y]).

?-  member(x,[a,b,c]).

?-  member(X,[a,b,c]).
```

Section 4.5

1. Write a 3-place predicate combine1 which takes three lists as arguments and combines the elements of the first two lists into the third as follows:

   ```
   ?-  combine1([a,b,c],[1,2,3],X).

   X  =  [a,1,b,2,c,3]

   ?-  combine1([f,b,yip,yup],[glu,gla,gli,glo],Result).

   Result  =  [f,glu,b,gla,yip,gli,yup,glo]
   ```

2. Now write a 3-place predicate combine2 which takes three lists as arguments and combines the elements of the first two lists into the third as follows:

   ```
   ?-  combine2([a,b,c],[1,2,3],X).

   X  =  [[a,1],[b,2],[c,3]]

   ?-  combine2([f,b,yip,yup],[glu,gla,gli,glo],Result).

   Result  =  [[f,glu],[b,gla],[yip,gli],[yup,glo]]
   ```

3. Finally, write a 3-place predicate combine3 which takes three lists as arguments and combines the elements of the first two lists into the third as follows:

   ```
   ?-  combine3([a,b,c],[1,2,3],X).

   X  =  [j(a,1),j(b,2),j(c,3)]

   ?-  combine3([f,b,yip,yup],[glu,gla,gli,glo],R).

   R  =  [j(f,glu),j(b,gla),j(yip,gli),j(yup,glo)]
   ```

Section 5.5 - exercises

Exercise 5.2

1. Define a 2-place predicate increment that holds only when its second argument is an integer one larger than its first argument. For example, increment(4,5) should hold, but increment(4,6) should not.
2. Define a 3-place predicate sum that holds only when its third argument is the sum of the first two arguments. For example,sum(4,5,9) should hold, but sum(4,6,12) should not.

Exercise 5.3 Write a predicate addone/2 whose first argument is a list of integers, and whose second argument is the list of integers obtained by adding 1 to each integer in the first list. For example, the query

```
?-  addone([1,2,7,2],X).
```

should give

```
X  =  [2,3,8,3].
```

Section 5.6 - tasks

2. In mathematics, an n-dimensional vector is a list of numbers of length n. For example, [2,5,12] is a 3-dimensional vector, and[45,27,3,-4,6] is a 5-dimensional vector. One of the basic operations on vectors is scalar multiplication . In this operation, every element of a vector is multiplied by some number. For example, if we scalar multiply the 3-dimensional vector [2,7,4] by 3 the result is the 3-dimensional vector [6,21,12] .

   Write a 3-place predicate scalarMult whose first argument is an integer, whose second argument is a list of integers, and whose third argument is the result of scalar multiplying the second argument by the first. For example, the query

   ```
   ?-  scalarMult(3,[2,7,4],Result).
   ```

   should yield

   ```
   Result  =  [6,21,12]
   ```

3. Another fundamental operation on vectors is the dot product . This operation combines two vectors of the same dimension and yields a number as a result. The operation is carried out as follows: the corresponding elements of the two vectors are multiplied, and the results added. For example, the dot product of [2,5,6] and [3,4,1] is 6+20+6 , that is, 32 . Write a 3-place predicate dotwhose first argument is a list of integers, whose second argument is a list of integers of the same length as the first, and whose third argument is the dot product of the first argument with the second. For example, the query

   ```
   ?-  dot([2,5,6],[3,4,1],Result).
   ```

   should yield

   ```
   Result  =  32
   ```

Section 6.3 - Exercises

Exercise 6.1 Let's call a list doubled if it is made of two consecutive blocks of elements that are exactly the same. For example,[a,b,c,a,b,c] is doubled (it's made up of [a,b,c] followed by [a,b,c]) and so is [foo,gubble,foo,gubble] . On the other hand,[foo,gubble,foo] is not doubled. Write a predicate doubled(List) which succeeds when List is a doubled list.

Exercise 6.2 A palindrome is a word or phrase that spells the same forwards and backwards. For example, ‘rotator’, ‘eve’, and ‘nurses run’ are all palindromes. Write a predicate palindrome(List) , which checks whether List is a palindrome. For example, to the queries

```
?- palindrome([r,o,t,a,t,o,r]).
```

and

```
?- palindrome([n,u,r,s,e,s,r,u,n]).
```

Prolog should respond yes, but to the query

```
?- palindrome([n,o,t,h,i,s]).
```

it should respond no.

Exercise 6.3 Write a predicate toptail(InList,OutList) which says no if InList is a list containing fewer than 2 elements, and which deletes the first and the last elements of InList and returns the result as OutList , when InList is a list containing at least 2 elements. For example:

```
toptail([a],T).
no

toptail([a,b],T).
T=[]

toptail([a,b,c],T).
T=[b]
```

(Hint: here’s where append/3 comes in useful.)

Exercise 6.4 Write a predicate last(List,X) which is true only when List is a list that contains at least one element and X is the last element of that list. Do this in two different ways:

1. Define last/2 using the predicate rev/2 discussed in the text.
2. Define last/2 using recursion.

Exercise 6.5 Write a predicate swapfl(List1,List2) which checks whether List1 is identical to List2 , except that the first and last elements are exchanged. Here’s where append/3 could come in useful again, but it is also possible to write a recursive definition without appealing to append/3 (or any other) predicates.

Exercise 6.6 Here is an exercise for those of you who like logic puzzles.

There is a street with three neighbouring houses that all have a different colour, namely red, blue, and green. People of different nationalities live in the different houses and they all have a different pet. Here are some more facts about them:

- The Englishman lives in the red house.
- The jaguar is the pet of the Spanish family.
- The Japanese lives to the right of the snail keeper.
- The snail keeper lives to the left of the blue house.

Who keeps the zebra? Don’t work it out for yourself: define a predicate zebra/1 that tells you the nationality of the owner of the zebra!

(Hint: Think of a representation for the houses and the street. Code the four constraints in Prolog. You may find member/2 and sublist/2useful.)

Section 6.4 - tasks

2. Write a predicate set(InList,OutList) which takes as input an arbitrary list, and returns a list in which each element of the input list appears only once. For example, the query

   ```
   set([2,2,foo,1,foo,  [],[]],X).
   ```

   should yield the result

   ```
   X  =  [2,foo,1,[]].
   ```

   (Hint: use the member predicate to test for repetitions of items you have already found.)

3. We 'flatten' a list by removing all the square brackets around any lists it contains as elements, and around any lists that its elements contain as elements, and so on, for all nested lists. For example, when we flatten the list

   ```
   [a,b,[c,d],[[1,2]],foo]
   ```

   we get the list

   ```
   [a,b,c,d,1,2,foo]
   ```

   and when we flatten the list

   ```
   [a,b,[[[[[[c,d]]]]]],[[1,2]],foo,[]]
   ```

   we also get

   ```
   [a,b,c,d,1,2,foo].
   ```

   Write a predicate flatten(List,Flat) that holds when the first argument List flattens to the second argument Flat . This should be done without making use of append/3 .

Ok, we're now halfway through the book. And flattening a list is the Pons Asinorum of Prolog programming. Did you cross it ok? If so, great. Time to quit.

Exercise

Exercise

Exercise

Exercise

Exercise