Assignment 3

02156 Logical Systems and Logic Programming, Fall 2023

November 12, 2023

Thomas Schiøler Hansen   s214968

1 Table of contents

Contents

1 Table of contents **2**

2 Problem 1 **3**

2.1 Problem 1.1 3

2.2 Problem 1.2 4

3 Problem 2 **5**

3.1 Problem 2.1 5

3.2 Problem 2.2 5

4 Problem 3 **6**

4.1 Problem 3.1 6

4.2 Problem 3.2 7

5 Problem 4 **8**

5.1 Problem 4.1 8

5.2 Problem 4.2 9

2 Problem 1

For all prolog code please see the attached prolog file.

2.1 Problem 1.1

In this problem I decided to use the findall predicate and search for the names within the list of students. I then used the sort predicate to sort the names to ensure that no duplicates were shown. When using the same query as in the problem description I then got the desired output. Please see it below.

```
?- students(S).
S = [alice, bruce, carol, dorit, erica, james, peter, xenia].
```

To test the predicate even further I decided to put a duplicate entry in the student "database" to see if it was able to merge the duplicates. I added the following:
score(exam, alice, 85).
Please see the output below:

```
?- students(S).
S = [alice, bruce, carol, dorit, erica, james, peter, xenia].
```

From this output we can see that alice did not feature in the list more than once even though she was in the database more than once. This showcases that the program can merge duplicates effectively.

If we instead try to use a variable, we could try the following.

```
?- students(_).
true.
```

This simply tests for if there is anything in the database, which there is. So it returns true as it should.
Next up we could try to test if it works for a partially instantiated list. We could do this by

moving alice in the head and T in the tail. The program should then only succeed if alice is in the database and return the rest of the names from the list.

```
?- students([alice—T]).
T = [bruce, carol, dorit, erica, james, peter, xenia].
```

2.2 Problem 1.2

In this solution I again decided to use the findall predicate in order to get all the students from the list that had a score above 40. I then took the length of the list that it created in order to know how many students fit the criteria and then multiplied the length by 1000 as each student would get 1000.

```
?- money(M).
M = 5000.
```

The query above returns 5000 as expected because five students had a score above 40.
Next I modified the list so no students had an exam score of over 40 to test if it would return 0 as expected. Please see the output below:

```
?- money(M).
M = 0.
```

Again a test to see if there are any students that fit the criteria can be seen below.

```
?- money(_).
true.
```

Here it returns true as there is more than one student that has an exam score of over 40, which is what we expected.
Lastly, we could test to see how much money the students would collect in total. As we know from a previous query the expected amount is 5000. So if we test to see if they would get 3000 it should return false as there are not only 3 students that satisfy the criteria.

?- money(3000).
false.

3 Problem 2

To complete these refutation and resolution proofs I have followed the four steps listed in the book.

3.1 Problem 2.1

Below please see the steps taken to reach a using refutation and resolution on the following formula: $(p \wedge q) \rightarrow (q \wedge p)$.

$$\begin{aligned}
\neg((p \wedge q) \rightarrow (q \wedge p)) &\equiv \neg(\neg(p \wedge q) \vee (q \wedge p)) \\
&\equiv \neg\neg(\neg\neg(p \wedge q) \wedge \neg(q \wedge p) \\
&\equiv (p \wedge q) \wedge \neg(q \wedge p) \\
&\equiv (p \wedge q) \wedge (\neg q \vee \neg p)
\end{aligned}$$

We can find the clauses below.

{(1)p (2)q, (3)¬q ¬p}
4. $\neg q$ 1,3
5. $\square$ 2,4
The clauses that were clashing have been resolved, hence the formula is valid.

3.2 Problem 2.2

I again followed the four steps to find out whether the formula $\neg((\neg p \vee \neg q) \wedge (p \vee q))$ is valid using refutation and resolution.

$$\begin{aligned}
\neg\neg((\neg p \vee \neg q) \wedge (p \vee q)) &\equiv \neg(\neg(\neg p \vee \neg q) \vee \neg(p \vee q)) \\
&\equiv \neg\neg(\neg p \vee \neg q) \wedge \neg\neg(p \vee q) \\
&\equiv (\neg p \vee \neg q) \wedge (p \vee q)
\end{aligned}$$

We can find the clauses below.

{(1)¬ p ¬ q, (2)p q}

3. □ 1,2

The clauses that were clashing have been resolved, hence the formula is valid.

4 Problem 3

For all prolog code please see the attached prolog file.

4.1 Problem 3.1

To solve this problem I created a member1(+List) program that splits the list up into a head and a tail. In order to see whether the head is in the rest of the list we simply use the member predicate to check if the head is in the tail of the list.
I have tested the program using all the sample queries to see if it works and it had successfully completed all those tests.
I could also try to test the program if there are multiple occurrences in the list of the head as seen below.

```
?- member1([2,5,2,2,3]).
true.
```

This returns true as expected because 2 is also in the tail of the list.
I now want to test the program using variables. An example of this is seen below.

```
?- member1([X, 2, X]).
X = 2.
```

It here correctly returns X=2 as the head then would be 2 and 2 is also in the tail meaning that it is correct.
Lastly, I want to try testing the program using an entire variable list. Here I would expect the output to be a pattern based on the implementation of my program. Please see the test below.

```
?- member1(L).
L = [_A, _A|_].
```

The output here is a representation of a given list with at least two elements. The two elements are the same which is indicated by the *_A*. The rest of the list does not matter, which is why an underscore is used. The rest could be no elements or many other elements.

4.2 Problem 3.2

Here I solved the problem be iterating over the head and tail of the lists recursively and handling the head and tails separately. The results are then combined into a list that is then sorted in order to avoid duplicates. If an element is not a list it is simply kept as it is.
I will now try to test the program with nested lists as seen below.

```
?- p([a, [b, c], [d, [e]], f], Result).
Result = [a, b, c, d, e, f].
```

Here we can see that it correctly sorts the nested lists.
Another test could be the base case when a list is empty. Please see the query below.

```
?- p([], Result).
Result = [].
```

It here correctly returns an empty list. We can also try to test the case with a list with a single element as seen below.

```
?- p([x], Result).
Result = [x].
```

It here again, correctly returns that single element.
I will now test using some variables to see if the program returns correct outputs.
Below is a test using X and Y variables as well as a small list.

```
?- p([X, [a, b], Y], Result).
X = Y, Y = [],
Result = [a, b].
```

From the output we can see that X and Y are treated as empty lists. This is because prolog wants to find the simplest solution for the program and since there is not additional information to what X and Y are, then it just puts them as empty lists. The result is therefore just the list [a,b].
In the output below there is another case where prolog unifies the variable with an empty list. This is like before due to the case "p([], []) :- !." in my program. Since prolog does not have any information on the variable it can easily unify the variable with an empty list to satisfy the predicate.

```
?- p(L, Result).
L = Result, Result = [].
```

5 Problem 4

For all prolog code please see the attached prolog file.

5.1 Problem 4.1

In this solution I again used the findall predicate to find all the words with a sortorder greater or equal to 100 and world class that is n or v. Afterwards the list created is then sorted to ensure that no duplicates will be displayed. The description of the problem states that there are 20 words that fit the criteria so to find out whether the program works correctly I ensured that all 20 words were printed. This was the case as can be seen below.

```
?- selectlist(List);true.
List = [be, come, do, find, get, give, go, have, know|...] [write]
List = [be, come, do, find, get, give, go, have, know, look, make, people, say, see, take, think,
time, use, way, year]
```

To ensure that the output list also could be empty I decided to set the findall sortorder to

¡1 so that no words would match. The output was an empty list as expected:

```
?- selectlist(List).
List = [] ;
true.
```

Next up I tried to test to see if it would still succeed if it was not bound to any specific list, because in that case it should still succeed.

```
?- selectlist(_).
true.
```

As seen above it did return true as expected.

5.2 Problem 4.2

To create the dump predicate I first find a word with the wordclass a and with the wordclass adv and a third wordclass as well. To make sure that the third wordclass is not a or adv we check it before printing the word and its third wordclass.
In addition, I have tested that the dump predicate successfully prints all 29 lines of words with their third wordclass which indicates that the program works flawlessly.