Gabbi Forsythe

Dr. Rybarczyk

CS441

22 November 2019

Assignment #5 Report: Prolog

#1 – TESTING AND IMPLEMENTATION (ans1.pl)

The first prolog predicate that I wrote for this project was a predicate for a family tree of sorts. For my implementation of this project, I only included female family members so that I didn't have to include brother/father/husband relationships. Doing the project this way kept the project simple and clear. However, my solution could easily be adapted to account for males. A `male(X)` and `female(X)` fact would need to be added for each person, and then would need to be added to the aunt/mother/sister rules. For my implementation of this predicate, I put all the people in the program based on who their mother is. Using this information, we can determine who is who's sister and therefore, who is the aunt of who.

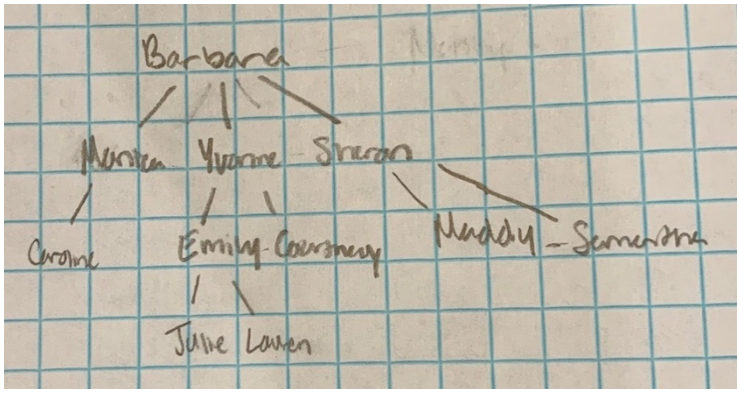


Photo 1: Family tree of everyone in this prolog predicate

As far as the testing of this program goes, I had four test cases that I wanted to go through. For my first test case, I wanted to test by putting two names (no variables) into the

query and ensure that I received the expected output. I entered in "`aunt(monica, caroline).`" and received the expected output of "`no`". For my second test case, I wanted to see if I could query for all the aunts of a given person. This query was written as "`aunt(X, caroline).`" and received the expected output (Yvonne and Sharon) as seen in the image below. The third test case that I tried was the opposite of the second test case. This time I wrote a query to find out all the people someone is an aunt of. The query I gave was "`aunt(monica, X).`" The output that I received was what was expected (Emily, Courtney, Maddy, and Samantha). The last test case I wanted to do was to run a query to find the aunt of someone who doesn't have any aunts. For this query I input "`aunt(X, barbara).`" and received the expected output of "`no`". These test cases provided complete coverage of my code because it covers one of each possible query that could be written for this particular predicate, and therefore provides complete coverage.

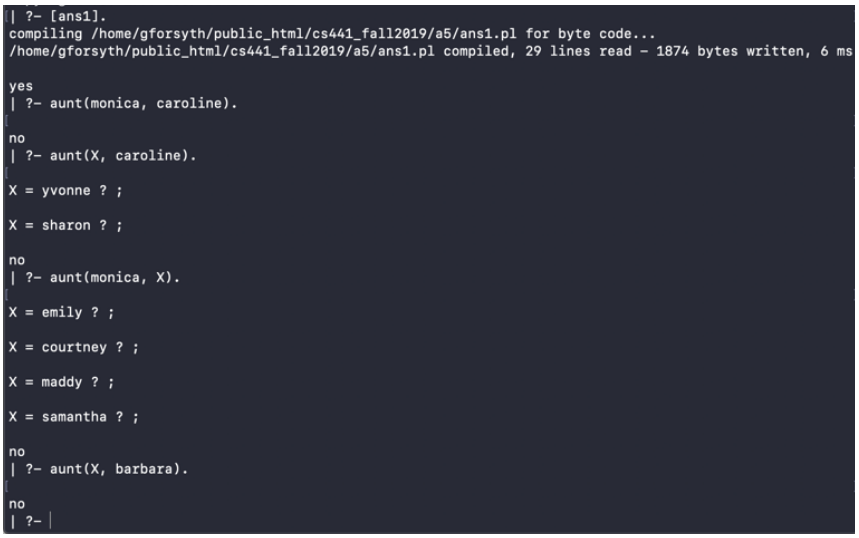
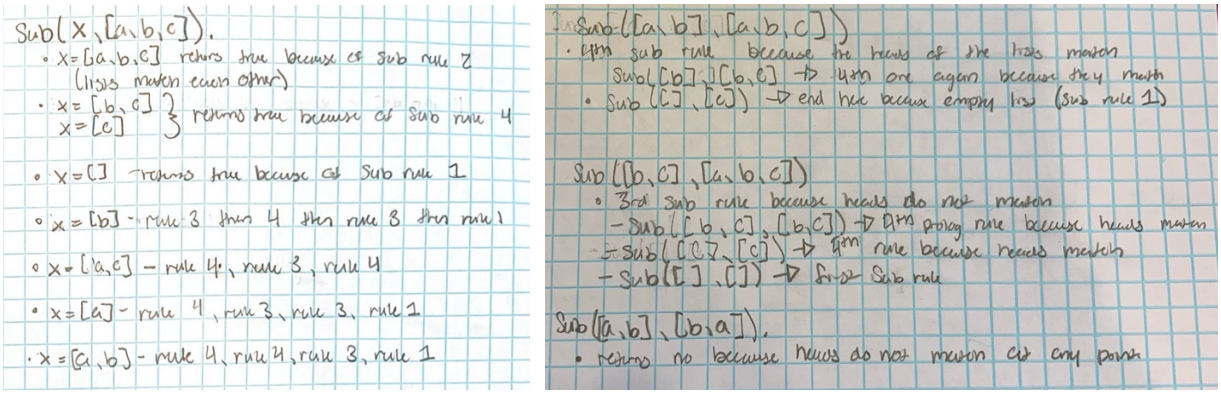


Photo 2: Results of the test cases for the first predicate

#2 – TESTING AND IMPLEMENTATION (ans2.pl)

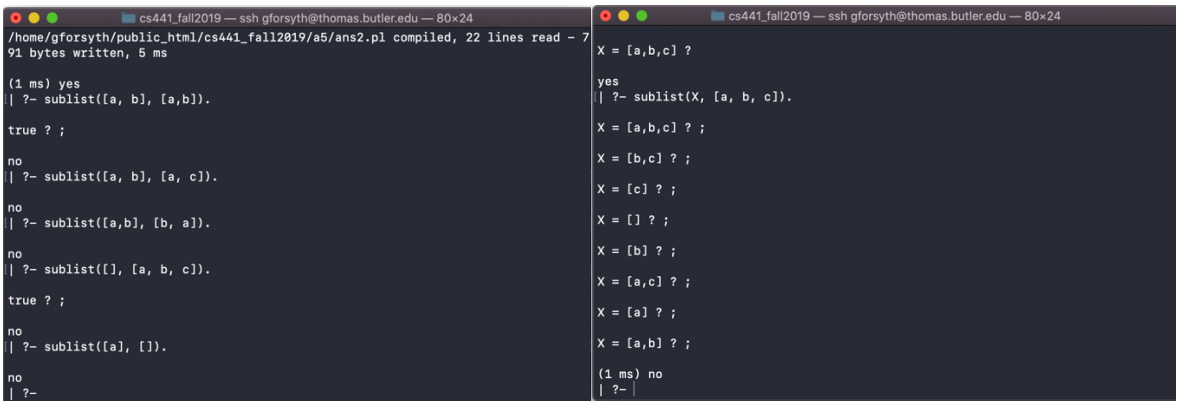
This prolog predicate determines if list X is a sublist of list Y. I called it `sub` because `sublist` conflicted with a built-in predicate in GNU Prolog. This predicate and I did not get

along. I worked on it for a couple of hours before turning to the internet for help. After finding a couple of different solutions to the problem (listed at end of report), I decided to write out the steps prolog was taking for each query and figure out what exactly was happening (seen in Photos #3 below). Writing out these steps helped me figure out how the solutions online was working and helped me formulate my own solution (which is obviously very similar to the ones I saw online).



Photos 3: Steps of Sublist Predicate

After formulating the solution to this predicate, it was time to test it and make sure that I actually had figured it out. I ran the same tests as were in the example in the assignment description (A5.pdf). Some of the results were not as expected. For example in the first image (of Photos #4), when the query said "`sublist([a, b], [a, b).`" and Prolog said "`true ? `", as if Prolog finds itself as confusing as I do. I tried to fix this issue by adding in cuts (`!`) in the different rules, however, it did not fix the problem. I don't think this was an issue, because when entering in other queries such as "`sublist([a,b], [b, a]).`" returned the expected answer of "`no`". All the other tests that I ran returned the expected answers, sometimes in different orders, however, all the information was present and therefore, it is a working predicate.



Photos 4: Results of Testing for `Ans2.pl`

#3 – TESTING AND IMPLEMENTATION (ans3.pl)

This prolog predicate uses the insertion sort algorithm to sort lists of integers. The implementation of this algorithm uses recursion to insert the numbers into the list. There are basically two lists happening in this algorithm, `PartiallySorted` and `Sorted`. As the algorithm goes through the unsorted list, it inserts the numbers into the list using the `insertion` rule, which compares the values of the list. Leaving us with a fully sorted list. To be honest, I used the bubble sort algorithm from class and added to it until it worked again. Then I sat down and drew it out to figure out why it worked, as seen below.

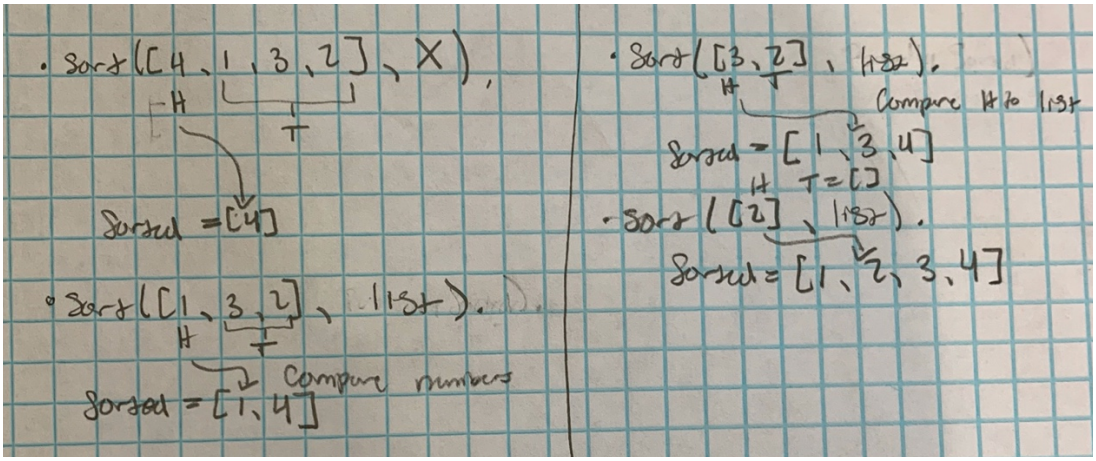


Photo 5: figuring out what is happening with insertion sort

As far as testing goes, I tested it on a couple different unsorted lists, one with only positive numbers, one with some negative numbers, and one with some duplicate values. The lists with positive numbers and the list that contained negative numbers provided the expected output of a sorted list. The list with duplicate values returned a sorted list with only one of every value. While this was not necessarily what I expected to see, it is not necessarily a bad output, so I will allow it. I wrote this program with the assumption that the list to be sorted would only contain integers, because that is what was given in the example in `A5.pdf`. Therefore, these tests provide sufficient coverage to my program.

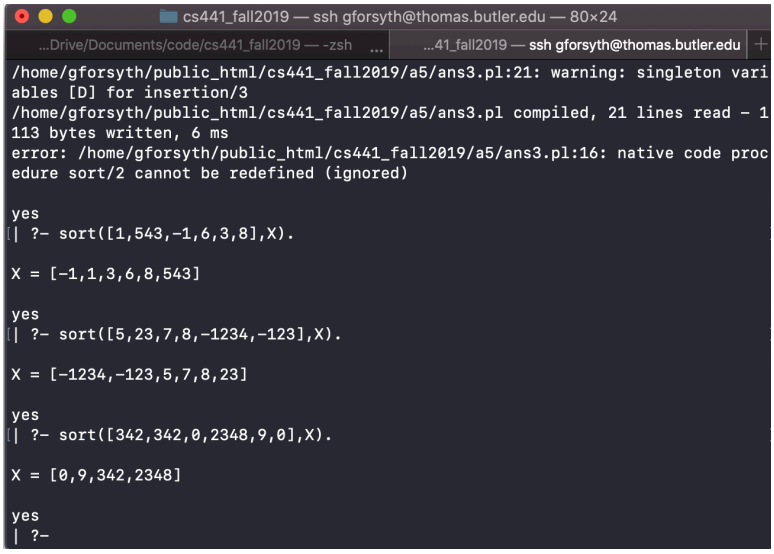


Photo 6: Results of the test cases for `ans3.pl`

THOUGHTS ON PROLOG

This week I have struggled with Prolog for many, many hours. Needless to say, I dislike it. However, I do see a lot of benefits to having to do this project. First off, it forced me to look at problems from a different perspective than normal, which is always good practice for programming and computer science in general. It also helped me get more practice with recursion, which is something I normally avoid at all costs. I had never used a declarative

programming language before, so this project gave me experience doing something new which always brings a lot of learning with it.

Prolog had a lot of similarities to the imperative languages that I have used before. The concepts involved with recursion and lists were similar to recursion and arrays in C++ and Java. However, the closed-world assumption and the idea of every line having its own variables (variables are not the same from line to line) were hard to adjust to. All in all, using Prolog was a cool experience, but I really hope I never have to use it again.

RESOURCES

https://stackoverflow.com/questions/7051400/prolog-first-list-is-sublist-of-second-list

https://stackoverflow.com/questions/42455589/how-to-get-all-consecutive-sublists-subsets-in-prolog

Bubble Sort slide of Dr. Rybarczyk's Lecture 30 Power Point