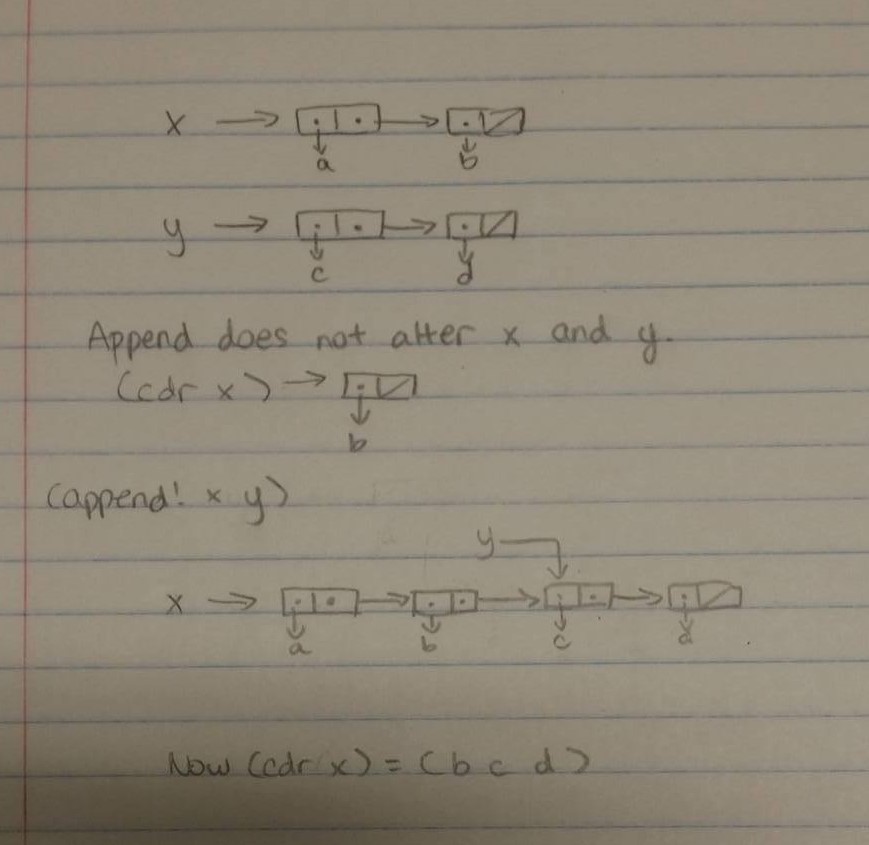
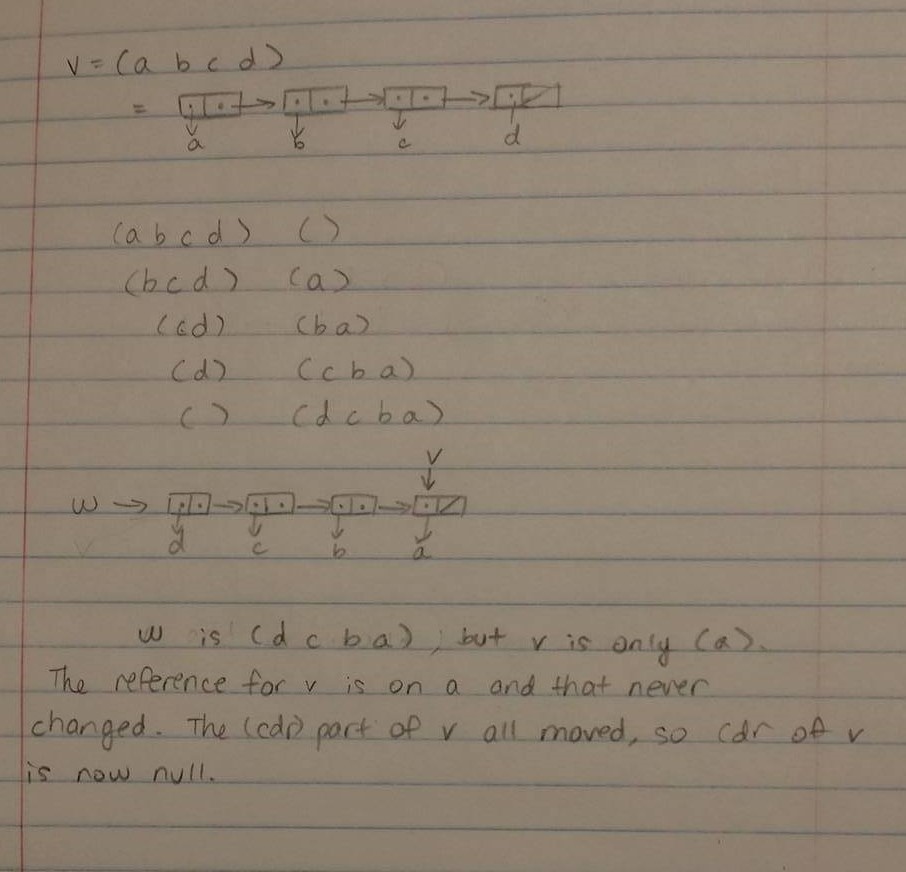
Kelvin Mei

COSI121B

PS3 Write-up

3.12



3.14

3.17

For this count-pairs, I used a second list y. y stores all the lists visited before, so when I’m counting pairs, I just check that with the ones in the lists and if it’s indeed in that then it doesn’t count the current pair. The comparison is done using memq, which takes the first occurrence of the element in the list and everything after it.

Lab:

0:

The cases for the marriages are separate because if a person received one proposal, then the most that would be changed is the proposer, the proposee’s fiancée and the proposee.

1: Send takes a list as parameter and sends a message to the objects in the list.

It’s a simple recursive function.

Courtship is a similar recursive method, taking in three parameters, the unengaged, proposers, and proposees. The proposees are not necessary, since the proposers have a preference list of the proposees and the proposee parameter is never used.

Couple? checks the intended for both parameters and then returns true if they match each other.

Currently-unengaged accepts a list of people and uses filter on the method unengaged? to filter out the unengaged from the engaged in the proposer list.

i-like-more? takes in two people and reports out a Boolean, true if the first person is more liked and false otherwise. This uses (length list) and (memq item list) to find the position of the person on the list. It also has a lambda procedure that accepts a list so the object can pass the preference list to this method.

2: ‘i-love-you has a lambda(asker) that takes into account whoever is asking. If the person is unengaged, then it will respond i-love-you-too and current-intended becomes the asker. Then if the person is engaged, it will call i-like-more? on the two contestants and pick the one they like more. If breaking up is needed, the acceptor will do the same thing as unengaged. Otherwise, it will respond buzz-off-creep.

3: Using write-line from problem set 2, I simply inserted them into wherever needed to print out the proposal/acceptance/dumping actions.

4: Assume there is an equal number of men and women so that no one is left unmarried. Note that one person’s preference list must have every one of the opposite gender on the list. Now assume there is one person that remains unmarried. The algorithm is set to run until there everyone is engaged, so if one person is unmarried, one other person of the other gender also has to be unmarried. Since the preference list must have that person of the opposite gender, by the pigeonhole principle, the two remaining couples can be matched and married. Since that means there is no one that remains unmarried, the original assumption was wrong by contradiction and everyone has to be paired.

5: For proposer P, every proposal is made from his/her preference list. The car of the list is taken first and proposals are repeated on the cdr of the list. That way, every consecutive proposal is to a person lower on the list, which means that person is liked less.

Now for acceptor A, every acceptance is also made from his/her preference list. The position of the proposer and the currently-intended are compared and the person higher on the list is chosen, so the person accepted has to be liked more by the acceptor.

6. Let m and w be a man and a woman, respectively that are not matched with each other. Assume m is the one who proposed. w must be on m’s preference list. Based on the proof of number 4, m must be married after the algorithm. So m may not be married to w because he was matched with a woman above w on the list or w rejected him and he married a woman below w on the list. Now m would not run off with w because 1) why would he run off with a woman that is lower on his preference list? and 2) w would not run off with someone she rejected because she chose a more favorable man over her. This works the same way if w was the proposer and m was the acceptor.

7: When (match-make) is called and the parameters are swapped, the results end up differently. The ones proposing gets their first choice while the acceptors get their second choice. This is because although the second proposer is more favored by the first acceptor, he never gets a chance to propose to that acceptor because the other acceptor has already accepted his proposal. Based on this, I would choose to be a proposer.