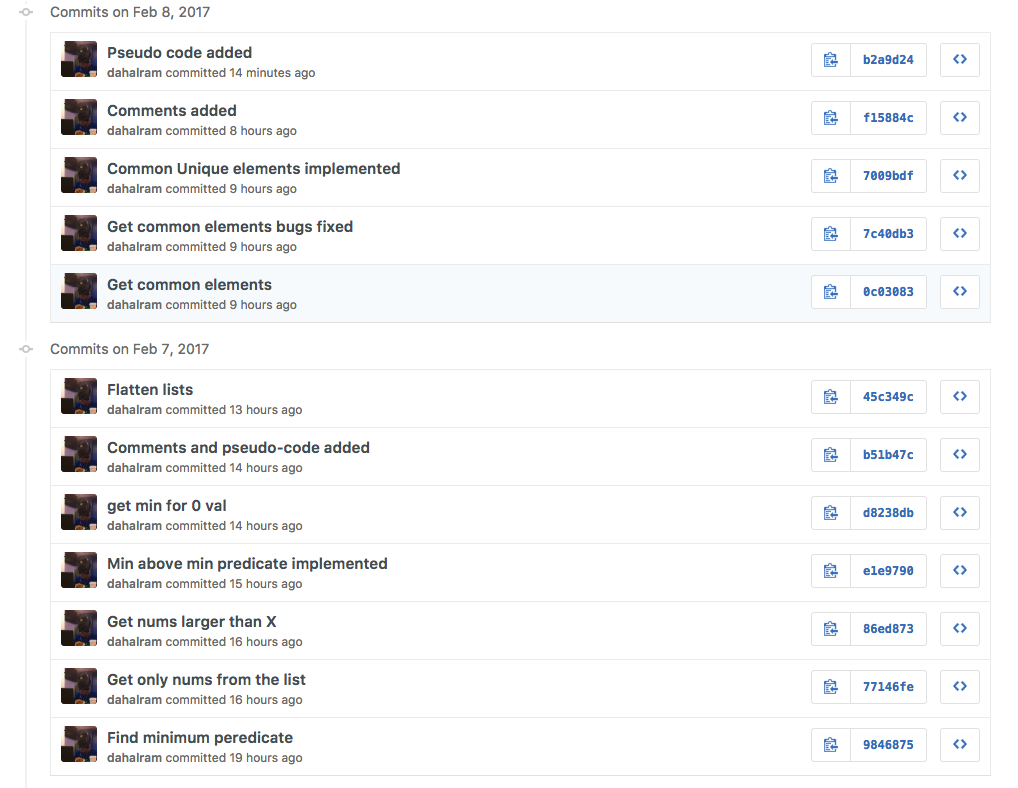
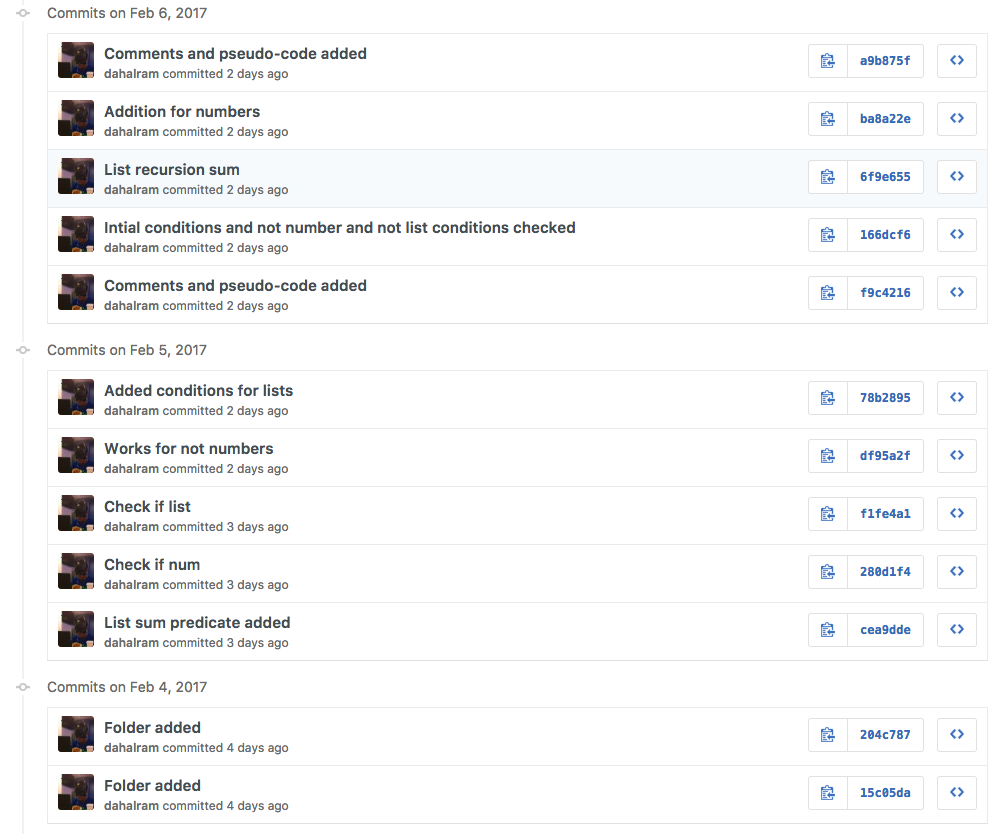
**Logic Programming Assignment**

Github URL: <https://github.com/dahalram/Programming-Languages/tree/master/Logic%20Programming%20Assignment>

The URL to the project: <https://github.com/dahalram/Programming-Languages.git>

Screenshots:





**Problem 1:**

Commit 1:

|  |
| --- |
| + |
|  |  | +% 1 sum-up-numbers-simple(L, N). |
|  |  | +# sum-up-numbers-simple(L, N). |
|  |  | + |
|  |  | +# sum-up-numbers-simple(L, N) :- |
|  |  | + |
|  |  | +list\_sum([], 0). |
|  |  | +list\_sum([Head | Tail], Sum) :- |
|  |  | + list\_sum(Tail, RestSum), |
|  |  | + Sum is Head + RestSum. |

Commit 2:

|  |
| --- |
| # sum-up-numbers-simple(L, N) :- |
|  |  |  |
|  |  | +is\_number(X, 0) :- |
|  |  | + not(number(X)). |
|  |  | + |
|  |  | list\_sum([], 0). |
|  |  | list\_sum([Head | Tail], Sum) :- |
|  |  | list\_sum(Tail, RestSum), |
|  |  | - Sum is Head + RestSum. |
|  |  | + Sum is Head + RestSum. |
|  |  | + |

Commit 3:

|  |
| --- |
| # sum-up-numbers-simple(L, N) :- |
|  |  |  |
|  |  | +list\_check(X, 0) :- |
|  |  | + is\_list(X). |
|  |  | + |
|  |  | is\_number(X, 0) :- |
|  |  | not(number(X)). |

Commit 4:

|  |
| --- |
| % 1 sum-up-numbers-simple(L, N). |
|  |  | -# sum-up-numbers-simple(L, N). |
|  |  |  |
|  |  | -# sum-up-numbers-simple(L, N) :- |
|  |  |  |
|  |  | list\_check(X, 0) :- |
|  |  | is\_list(X). |
|  | | @@ -16,3 +14,9 @@ |
|  |  | list\_sum(Tail, RestSum), |
|  |  | Sum is 0 + RestSum. |
|  |  |  |
|  |  | +list\_sum([Head|Tail], Sum) :- |
|  |  | + number(Head), |
|  |  | + not(is\_list(Head)), |
|  |  | + list\_sum(Tail, RestSum), |
|  |  | + Sum is Head + RestSum. |
|  |  | + |

Commit 5:

|  |
| --- |
| % 1 sum-up-numbers-simple(L, N). |
|  |  |  |
|  |  | +% Pseudo-code |
|  |  | +% Check if the list is empty: return 0 |
|  |  | +% Check if the element is a number or not |
|  |  | +% Check if the element is a list or not |
|  |  |  |
|  |  | -list\_check(X, 0) :- |
|  |  | - is\_list(X). |
|  |  | +% Return 0 for empty list |
|  |  | +sum-up-numbers-simple([], 0). |
|  |  |  |
|  |  | -is\_number(X, 0) :- |
|  |  | - not(number(X)). |
|  |  | - |
|  |  | -list\_sum([], 0). |
|  |  | -list\_sum([Head | Tail], Sum) :- |
|  |  | +% If the element is not number then put it as 0 and recurse |
|  |  | +sum-up-numbers-simple([Head | Tail], Sum) :- |
|  |  | not(number(Head)), |
|  |  | - list\_sum(Tail, RestSum), |
|  |  | + sum-up-numbers-simple(Tail, RestSum), |
|  |  | Sum is 0 + RestSum. |
|  |  |  |
|  |  | -list\_sum([Head|Tail], Sum) :- |
|  |  | +% If the number is not zero and the element is a list, put it as 0 |
|  |  | +sum-up-numbers-simple([Head | Tail], Sum) :- |
|  |  | number(Head), |
|  |  | not(is\_list(Head)), |
|  |  | - list\_sum(Tail, RestSum), |
|  |  | + sum-up-numbers-simple(Tail, RestSum), |
|  |  | Sum is Head + RestSum. |

**Problem 2:**

Commit 1:

|  |
| --- |
| +%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% |
|  |  | +% 2 sum-up-numbers-general(L, N). |
|  |  | + |
|  |  | +sum-up-numbers-general([], 0). |
|  |  | + |
|  |  | +sum-up-numbers-general([Head | Tail], Sum) :- |
|  |  | + not(number(Head)), |
|  |  | + not(is\_list(Head)), |
|  |  | + sum-up-numbers-general(Tail, RestSum), |
|  |  | + Sum is 0 + RestSum. |
|  |  | + |

Commit 2:

|  |
| --- |
| sum-up-numbers-general([Head | Tail], Sum) :- |
|  |  | + not(number(Head)), |
|  |  | + is\_list(Head), |
|  |  | + sum-up-numbers-general(Head, HeadSum), |
|  |  | + sum-up-numbers-general(Tail, TailSum), |
|  |  | + Sum is HeadSum + TailSum. |
|  |  | + |

Commit 3:

|  |
| --- |
| sum-up-numbers-general([Head | Tail], Sum) :- |
|  |  | + number(Head), |
|  |  | + not(is\_list(Head)), |
|  |  | + sum-up-numbers-general(Tail, RestSum), |
|  |  | + Sum is Head + RestSum. |
|  |  | + |

Commit 4:

|  |
| --- |
| % 2 sum-up-numbers-general(L, N). |
|  |  |  |
|  |  | +% Pseudo-code |
|  |  | +% Check if the list is empty |
|  |  | +% Check if the element is a number |
|  |  | +% Check if the element is a list |
|  |  | +% Check if the element is neither a number nor a list |
|  |  | +% Add the sum and recurse according to what the element is. |
|  |  | + |
|  |  | +% Return 0 for empty list |
|  |  | sum-up-numbers-general([], 0). |
|  |  |  |
|  |  | +% If the element is not a number and not a list, then return 0 |
|  |  | sum-up-numbers-general([Head | Tail], Sum) :- |
|  |  | not(number(Head)), |
|  |  | not(is\_list(Head)), |
|  |  | sum-up-numbers-general(Tail, RestSum), |
|  |  | Sum is 0 + RestSum. |
|  |  |  |
|  |  | +% If the element is not a number but a list, do a recursion to get the sum from that list element |
|  |  | sum-up-numbers-general([Head | Tail], Sum) :- |
|  |  | not(number(Head)), |
|  |  | is\_list(Head), |
|  |  | sum-up-numbers-general(Head, HeadSum), |
|  |  | sum-up-numbers-general(Tail, TailSum), |
|  |  | Sum is HeadSum + TailSum. |
|  |  |  |
|  |  | +% If the element is just a number, add it to the sum |
|  |  | sum-up-numbers-general([Head | Tail], Sum) :- |
|  |  | number(Head), |
|  |  | not(is\_list(Head)), |

Commit 5:

Sorry only 4 commits for this problem.

**Problem 3:**

Commit 1:

|  |
| --- |
| %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% |
|  |  | +% 3 min-above-min(L1, L2, N). |
|  |  | + |
|  |  | +get\_min([X], X). |
|  |  | + |
|  |  | +get\_min([X, Y | Rest], Min) :- |
|  |  | + X < Y, |
|  |  | + get\_min([X | Rest], Min). |
|  |  | + |
|  |  | +get\_min([X, Y | Rest], Min) :- |
|  |  | + get\_min([Y | Rest], Min). |
|  |  | + |

Commit 2:

|  |
| --- |
| get\_nums([], []). |
|  |  | + |
|  |  | +get\_nums([Head | Rest], List) :- |
|  |  | + not(number(Head)), |
|  |  | + get\_nums(Rest, List). |
|  |  | + |
|  |  | +get\_nums([Head | Rest], [Head | List]) :- |
|  |  | + get\_nums(Rest, List). |
|  |  |  |

Commit 3:

|  |
| --- |
| get\_larger([], \_, []). |
|  |  |  |
|  |  | get\_larger([Head | Rest], X, List) :- |
|  |  | - Head > X, |
|  |  | + Head =< X, |
|  |  | get\_larger(Rest, X, List). |
|  |  |  |
|  |  | get\_larger([Head | Rest], X, [Head | List]) :- |
|  |  | get\_larger(Rest, X, List). |
|  |  |  |
|  |  |  |
|  |  | +min\_above\_min(L1, L2, N) :- |
|  |  | + get\_nums(L2, Res2), |
|  |  | + get\_nums(L1, Res1), |
|  |  | + get\_min(Res2, Min2), |
|  |  | + get\_larger(Res1, Min2, Larger), |
|  |  | + get\_min(Larger, N). |

Commit 4:

|  |
| --- |
| get\_min([X], X). |
|  |  | +get\_min([], 0). |
|  |  |  |
|  |  | get\_min([X, Y | Rest], Min) :- |
|  |  | X < Y, |
|  | | @@ -94,11 +95,13 @@ |
|  |  |  |
|  |  |  |
|  |  | min\_above\_min(L1, L2, N) :- |
|  |  | - get\_nums(L2, Res2), |
|  |  | - get\_nums(L1, Res1), |
|  |  | - get\_min(Res2, Min2), |
|  |  | - get\_larger(Res1, Min2, Larger), |
|  |  | + get\_nums(L2, Nums2), |
|  |  | + get\_min(Nums2, Min2), |
|  |  | + get\_nums(L1, Nums1), |
|  |  | + get\_larger(Nums1, Min2, Larger), |
|  |  | get\_min(Larger, N). |
|  |  | + %N is 0 + N1. |
|  |  | + |

Commit 5:

|  |
| --- |
| %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% |
|  |  | % 3 min-above-min(L1, L2, N). |
|  |  |  |
|  |  | +% Pseudo-code |
|  |  | +% First get the numbers from L1 and L2 |
|  |  | +% Then, get the minimum from L2, call it X |
|  |  | +% Get values larger than X in L1 |
|  |  | +% Return the smallest value from L1 greater than X |
|  |  | + |
|  |  | +% Get minimum value from a list |
|  |  | +% If empty, return 0, if single element, return that element |
|  |  | get\_min([X], X). |
|  |  | get\_min([], 0). |
|  |  |  |
|  | | @@ -74,6 +82,8 @@ |
|  |  | get\_min([Y | Rest], Min). |
|  |  |  |
|  |  |  |
|  |  | +% Get the numbers from the list |
|  |  | +% If empty list, return empty list |
|  |  | get\_nums([], []). |
|  |  |  |
|  |  | get\_nums([Head | Rest], List) :- |
|  | | @@ -83,7 +93,7 @@ |
|  |  | get\_nums([Head | Rest], [Head | List]) :- |
|  |  | get\_nums(Rest, List). |
|  |  |  |
|  |  | - |
|  |  | +% Get values larger than a given number (i.e. larger than the minimum from L2) |
|  |  | get\_larger([], \_, []). |
|  |  |  |
|  |  | get\_larger([Head | Rest], X, List) :- |
|  | | @@ -94,7 +104,7 @@ |
|  |  | get\_larger(Rest, X, List). |
|  |  |  |
|  |  |  |
|  |  | -min\_above\_min(L1, L2, N) :- |
|  |  | +min-above-min(L1, L2, N) :- |
|  |  | get\_nums(L2, Nums2), |
|  |  | get\_min(Nums2, Min2), |
|  |  | get\_nums(L1, Nums1), |

**Problem 4:**

Commit 1:

|  |
| --- |
| %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% |
|  |  | +% 4 common-unique-elements(L1,L2,N). |
|  |  |  |
|  |  | +flatten\_list([], []). |
|  |  |  |
|  |  | +flatten\_list([Head | Tail], Flat) :- |
|  |  | + flatten\_list(Head, Fl1), |
|  |  | + flatten\_list(Tail, Fl2), |
|  |  | + append(Fl1, Fl2, Flat). |
|  |  |  |
|  |  | - |
|  |  | - |
|  |  | +flatten\_list(Head, [Head]). |

Commit 2:

|  |
| --- |
| flatten\_list(Head, [Head]). |
|  |  |  |
|  |  | +get\_common\_elements([], \_, []). |
|  |  | + |
|  |  | +get\_common\_elements([Head | Tail], L2, [Head | Res]) :- |
|  |  | + member(Head, L2), |
|  |  | + get\_common\_elements(Tail, L2, Res). |
|  |  | + |

Commit 3:

|  |
| --- |
| +get\_common\_elements([\_|Tail], L2, Res) :- |
|  |  | + get\_common\_elements(Tail, L2, Res). |
|  |  |  |

Commit 4:

|  |
| --- |
|  |
| common-unique-elements(L1, L2, N) :- |
|  |  | + flatten\_list(L1, List1), |
|  |  | + flatten\_list(L2, List2), |
|  |  | + get\_common\_elements(List1, List2, N). |
|  |  |  |

Commit 5:

|  |
| --- |
| % 4 common-unique-elements(L1,L2,N). |
|  |  |  |
|  |  | +% Remove the nested and flatten the list |
|  |  | flatten\_list([], []). |
|  |  |  |
|  |  | flatten\_list([Head | Tail], Flat) :- |
|  | | @@ -124,6 +125,7 @@ |
|  |  |  |
|  |  | flatten\_list(Head, [Head]). |
|  |  |  |
|  |  | +% Get common elements from the two lists |
|  |  | get\_common\_elements([], \_, []). |
|  |  |  |
|  |  | get\_common\_elements([Head | Tail], L2, [Head | Res]) :- |
|  | | @@ -133,7 +135,7 @@ |
|  |  | get\_common\_elements([\_|Tail], L2, Res) :- |
|  |  | get\_common\_elements(Tail, L2, Res). |
|  |  |  |
|  |  | - |
|  |  | +% Get the common unique elements |
|  |  | common-unique-elements(L1, L2, N) :- |
|  |  | flatten\_list(L1, List1), |
|  |  | flatten\_list(L2, List2), |

Thank you,