Nanyang Technological University

CZ3005 AI
LAB GROUP TS9

LAB 2 REPORT

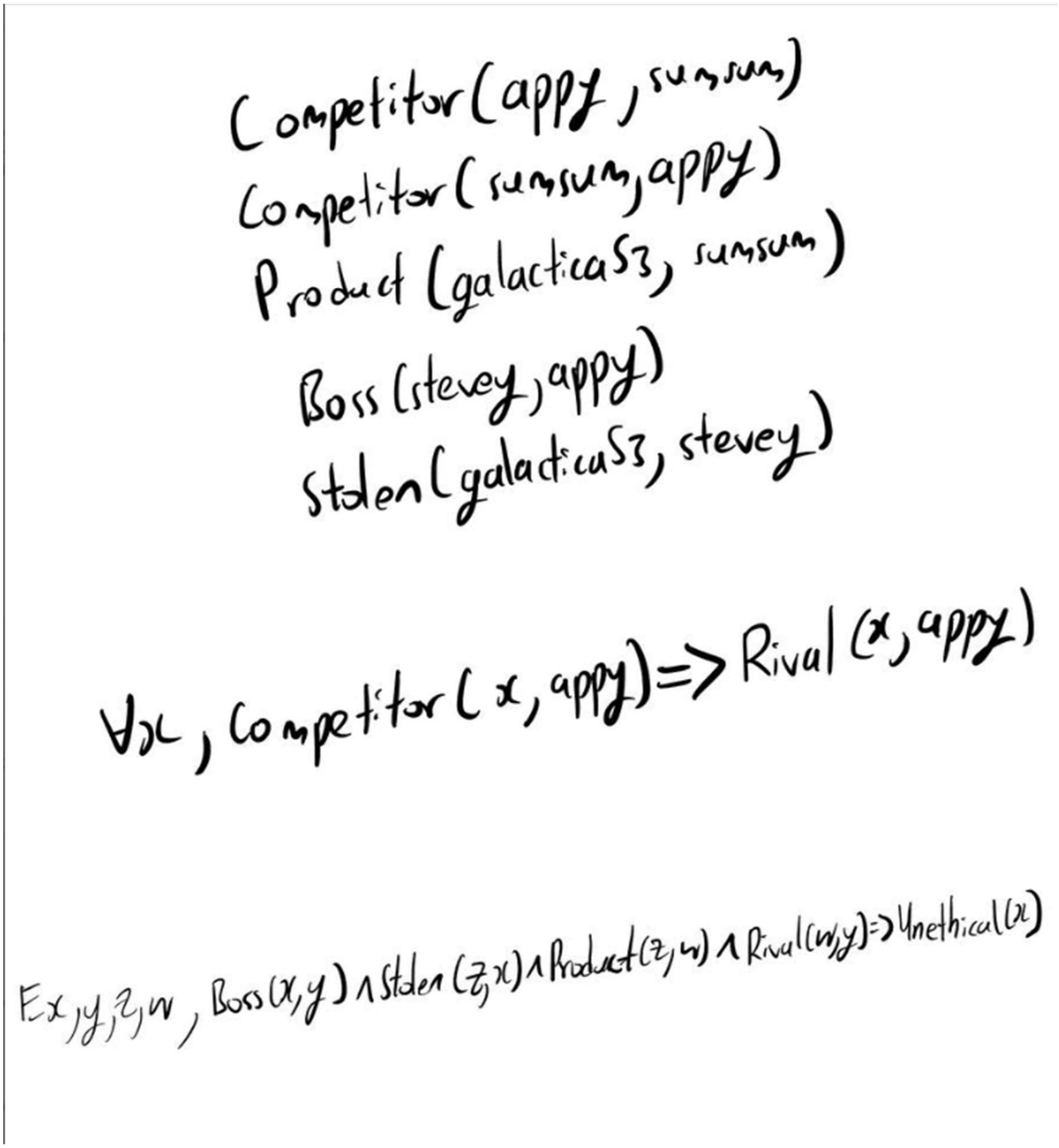
Prepared by:

Au Yew Rong Roydon (U2021424J)
Michelle Lam Su-Ann (U2021548K)
Low Lee Hang (U2021604G)

Contents

Exercise 1:

Qn 1:



Competitor(appy, sumsum)

Competitor(sumsum, appy)

Product(galacticaS3, sumsum)

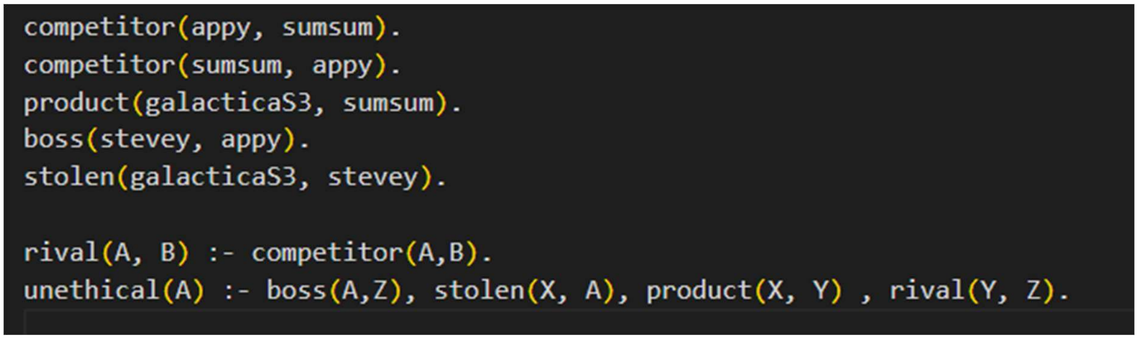
Boss(stevey, appy)

Stolen(galacticaS3, stevey)

$\forall x, \text{Competitor}(x, \text{appy}) \Rightarrow \text{Rival}(x, \text{appy})$

$E x, y, z, w, \text{Boss}(x, y) \land \text{Stolen}(z, x) \land \text{Product}(z, w) \land \text{Rival}(w, y) \Rightarrow \text{Unethical}(x)$

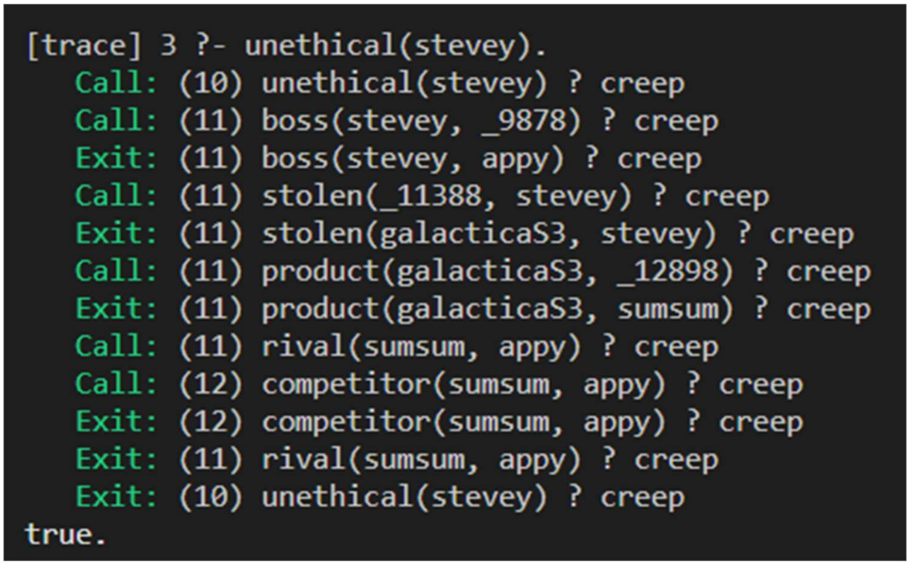
Qn 2:



```
competitor(appy, sumsum).
competitor(sumsum, appy).
product(galacticaS3, sumsum).
boss(stevey, appy).
stolen(galacticaS3, stevey).

rival(A, B) :- competitor(A,B).
unethical(A) :- boss(A,Z), stolen(X, A), product(X, Y) , rival(Y, Z).
```

Qn 3: Show stevey is unethical.

Result: True

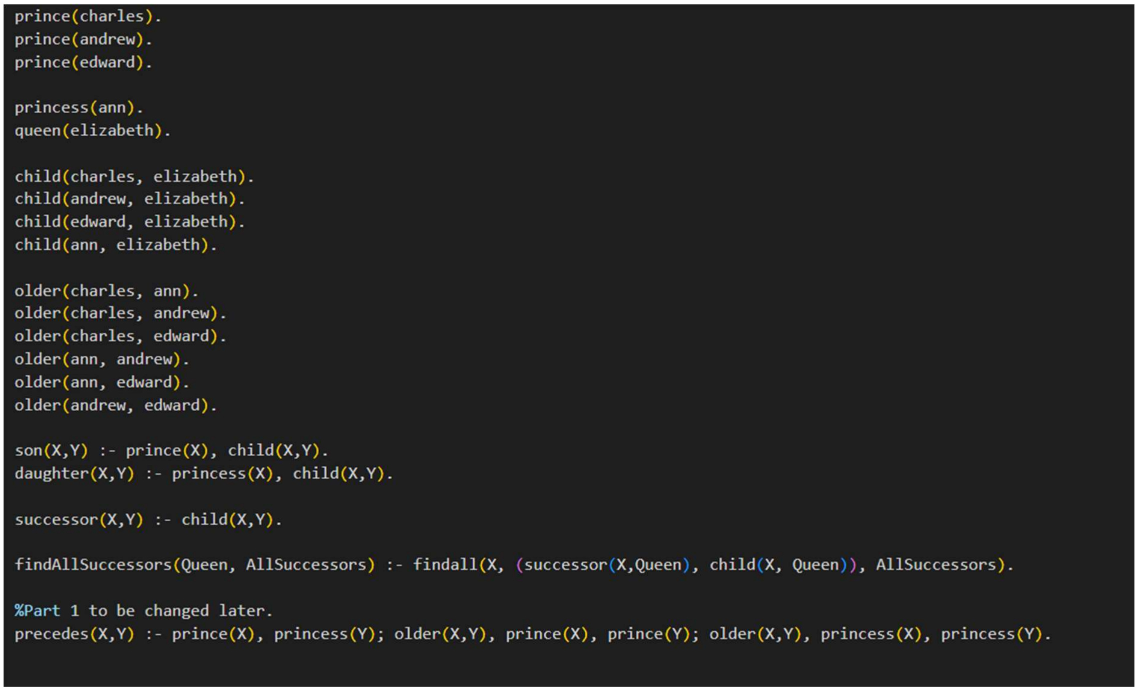


```
[trace] 3 ?- unethical(stevey).
   Call: (10) unethical(stevey) ? creep
   Call: (11) boss(stevey, _9878) ? creep
   Exit: (11) boss(stevey, appy) ? creep
   Call: (11) stolen(_11388, stevey) ? creep
   Exit: (11) stolen(galacticaS3, stevey) ? creep
   Call: (11) product(galacticaS3, _12898) ? creep
   Exit: (11) product(galacticaS3, sumsum) ? creep
   Call: (11) rival(sumsum, appy) ? creep
   Call: (12) competitor(sumsum, appy) ? creep
   Exit: (12) competitor(sumsum, appy) ? creep
   Exit: (11) rival(sumsum, appy) ? creep
   Exit: (10) unethical(stevey) ? creep
true.
```

Exercise 2:

Qn 1:

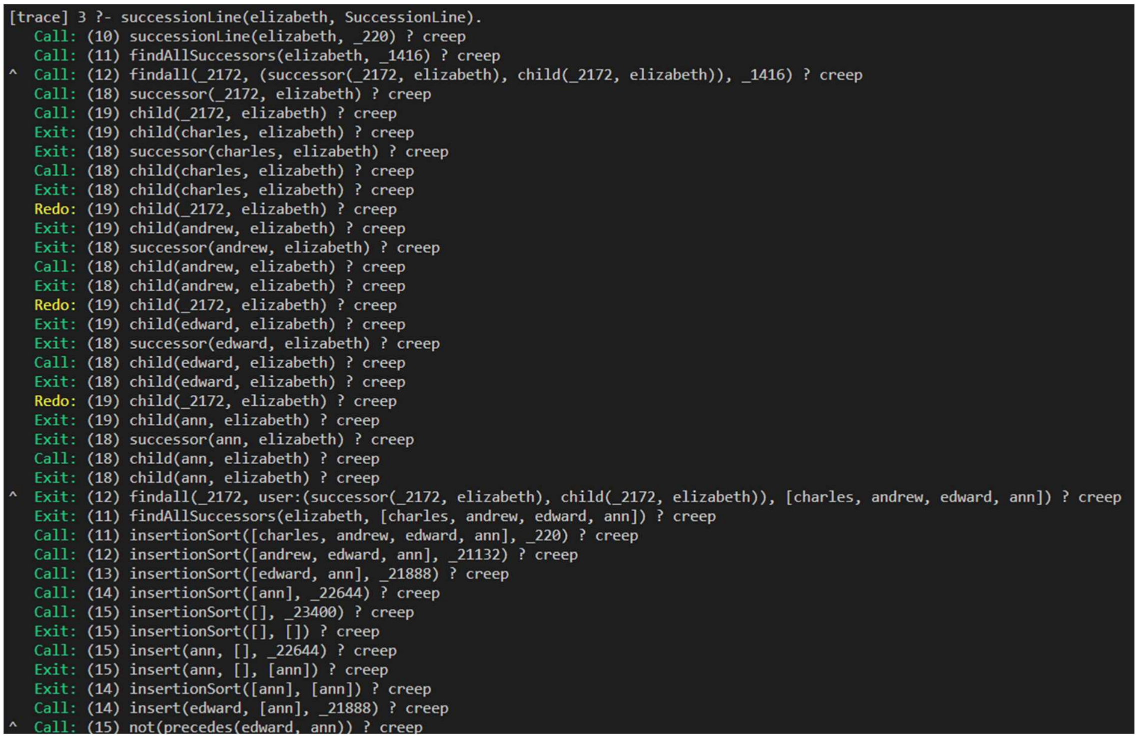
Relations and rules:



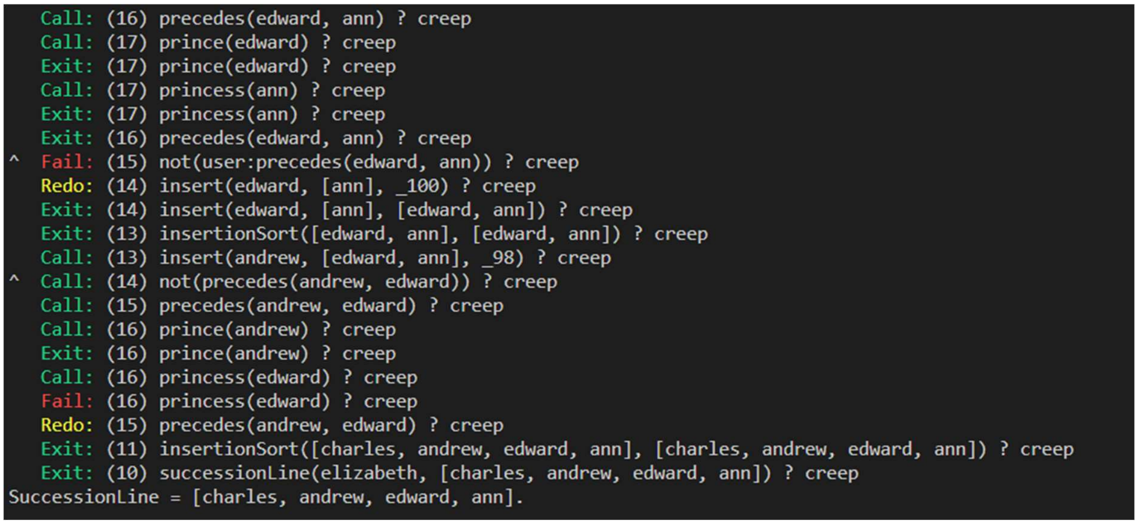
```
prince(charles).
prince(andrew).
prince(edward).

princess(ann).
queen(elizabeth).

child(charles, elizabeth).
child(andrew, elizabeth).
child(edward, elizabeth).
child(ann, elizabeth).

older(charles, ann).
older(charles, andrew).
older(charles, edward).
older(ann, andrew).
older(ann, edward).
older(andrew, edward).

son(X,Y) :- prince(X), child(X,Y).
daughter(X,Y) :- princess(X), child(X,Y).

successor(X,Y) :- child(X,Y).

findAllSuccessors(Queen, AllSuccessors) :- findall(X, (successor(X,Queen), child(X, Queen)), AllSuccessors).

%Part 1 to be changed later.
precedes(X,Y) :- prince(X), princess(Y); older(X,Y), prince(X), prince(Y); older(X,Y), princess(X), princess(Y).
```

```
%Sort using insertSort.

insertionSort([], []).
insertionSort([H|T], SuccessionLine) :-
  insertionSort(T, SortedTail),
  insert(H, SortedTail, SuccessionLine).

insert(A, [H|T], [H|Result]) :- not(precedes(A, H)), !, insert(A, T, Result).
insert(H, T, [H|T]).

successionLine(Queen, SuccessionLine) :-
    findAllSuccessors(Queen, AllSuccessors),
    insertionSort(AllSuccessors, SuccessionLine).


%Sorting using quicksort(alternate).
/*
sussessionListSort([],[]).
sussessionListSort([H|T], SuccessionLine) :-
    partition(H, T, Less, Greater),
    sussessionListSort(Less, SortedLess),
    sussessionListSort(Greater, SortedGreater),
    append(SortedLess, [H|SortedGreater], SuccessionLine).

partition(_,[],[],[]).
partition(P, [H|T], [H|Less], Greater) :-
    precedes(H,P),
    partition(P,T,Less,Greater).
partition(P, [H|T], Less, [H|Greater]) :-
    not(precedes(H,P)),
    partition(P,T,Less,Greater).

successionLine(Queen, SuccessionLine) :-
    findAllSuccessors(Queen, AllSuccessors),
    sussessionListSort(AllSuccessors, SuccessionLine).
*/
```

Result: [charles, andrew, edward, ann]



```
[trace] 3 ?- successionLine(elizabeth, SuccessionLine).
   Call: (10) successionLine(elizabeth, _220) ? creep
   Call: (11) findAllSuccessors(elizabeth, _1416) ? creep
^  Call: (12) findall(_2172, (successor(_2172, elizabeth), child(_2172, elizabeth)), _1416) ? creep
   Call: (18) successor(_2172, elizabeth) ? creep
   Call: (19) child(_2172, elizabeth) ? creep
   Exit: (19) child(charles, elizabeth) ? creep
   Exit: (18) successor(charles, elizabeth) ? creep
   Call: (18) child(charles, elizabeth) ? creep
   Exit: (18) child(charles, elizabeth) ? creep
   Redo: (19) child(_2172, elizabeth) ? creep
   Exit: (19) child(andrew, elizabeth) ? creep
   Exit: (18) successor(andrew, elizabeth) ? creep
   Call: (18) child(andrew, elizabeth) ? creep
   Exit: (18) child(andrew, elizabeth) ? creep
   Redo: (19) child(_2172, elizabeth) ? creep
   Exit: (19) child(edward, elizabeth) ? creep
   Exit: (18) successor(edward, elizabeth) ? creep
   Call: (18) child(edward, elizabeth) ? creep
   Exit: (18) child(edward, elizabeth) ? creep
   Redo: (19) child(_2172, elizabeth) ? creep
   Exit: (19) child(ann, elizabeth) ? creep
   Exit: (18) successor(ann, elizabeth) ? creep
   Call: (18) child(ann, elizabeth) ? creep
   Exit: (18) child(ann, elizabeth) ? creep
^  Exit: (12) findall(_2172, user:(successor(_2172, elizabeth), child(_2172, elizabeth)), [charles, andrew, edward, ann]) ? creep
   Exit: (11) findAllSuccessors(elizabeth, [charles, andrew, edward, ann]) ? creep
   Call: (11) insertionSort([charles, andrew, edward, ann], _220) ? creep
   Call: (12) insertionSort([andrew, edward, ann], _21132) ? creep
   Call: (13) insertionSort([edward, ann], _21888) ? creep
   Call: (14) insertionSort([ann], _22644) ? creep
   Call: (15) insertionSort([], _23400) ? creep
   Exit: (15) insertionSort([], []) ? creep
   Call: (15) insert(ann, [], _22644) ? creep
   Exit: (15) insert(ann, [], [ann]) ? creep
   Exit: (14) insertionSort([ann], [ann]) ? creep
   Call: (14) insert(edward, [ann], _21888) ? creep
^  Call: (15) not(precedes(edward, ann)) ? creep
```

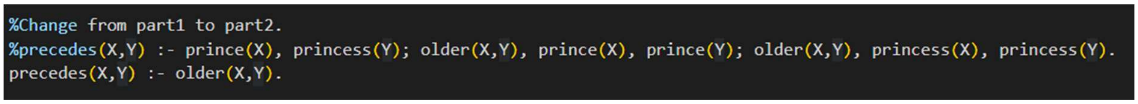


```
   Call: (16) precedes(edward, ann) ? creep
   Call: (17) prince(edward) ? creep
   Exit: (17) prince(edward) ? creep
   Call: (17) princess(ann) ? creep
   Exit: (17) princess(ann) ? creep
   Exit: (16) precedes(edward, ann) ? creep
^  Fail: (15) not(user:precedes(edward, ann)) ? creep
   Redo: (14) insert(edward, [ann], _100) ? creep
   Exit: (14) insert(edward, [ann], [edward, ann]) ? creep
   Exit: (13) insertionSort([edward, ann], [edward, ann]) ? creep
   Call: (13) insert(andrew, [edward, ann], _98) ? creep
^  Call: (14) not(precedes(andrew, edward)) ? creep
   Call: (15) precedes(andrew, edward) ? creep
   Call: (16) prince(andrew) ? creep
   Exit: (16) prince(andrew) ? creep
   Call: (16) princess(edward) ? creep
   Fail: (16) princess(edward) ? creep
   Redo: (15) precedes(andrew, edward) ? creep
   Exit: (11) insertionSort([charles, andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (10) successionLine(elizabeth, [charles, andrew, edward, ann]) ? creep
SuccessionLine = [charles, andrew, edward, ann].
```

Qn 2:

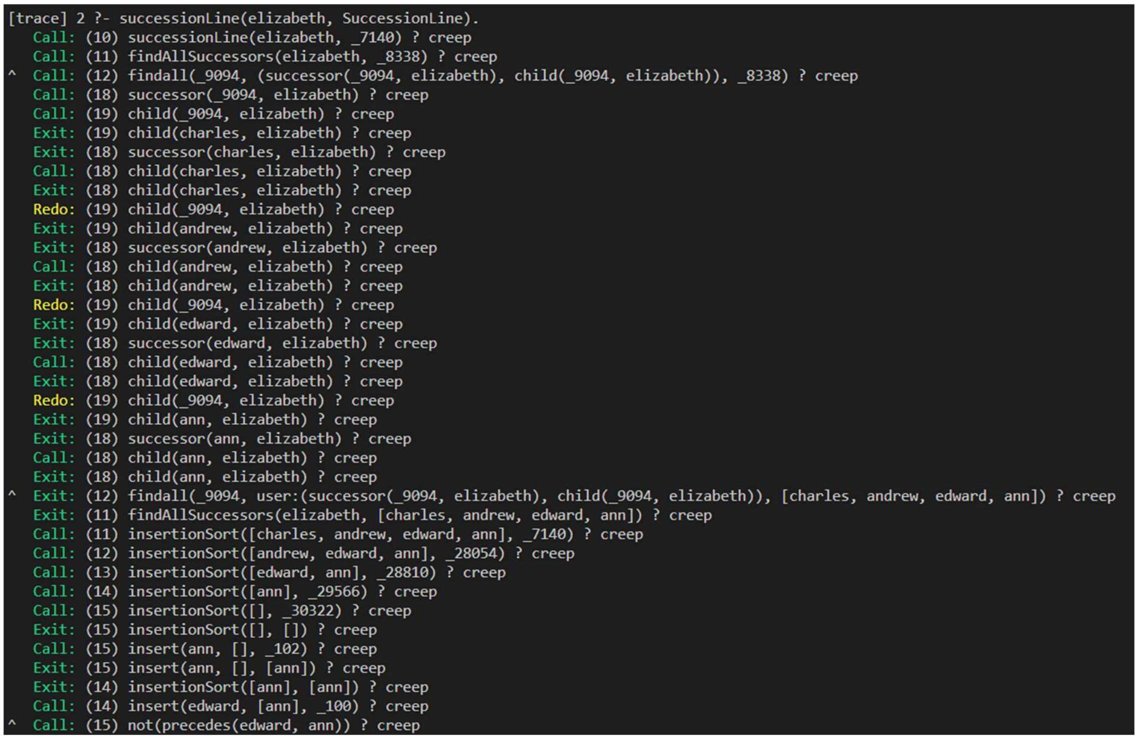
Result: [charles, ann, andrew, edward]

To achieve the result below we only changed one line of code. The line is:

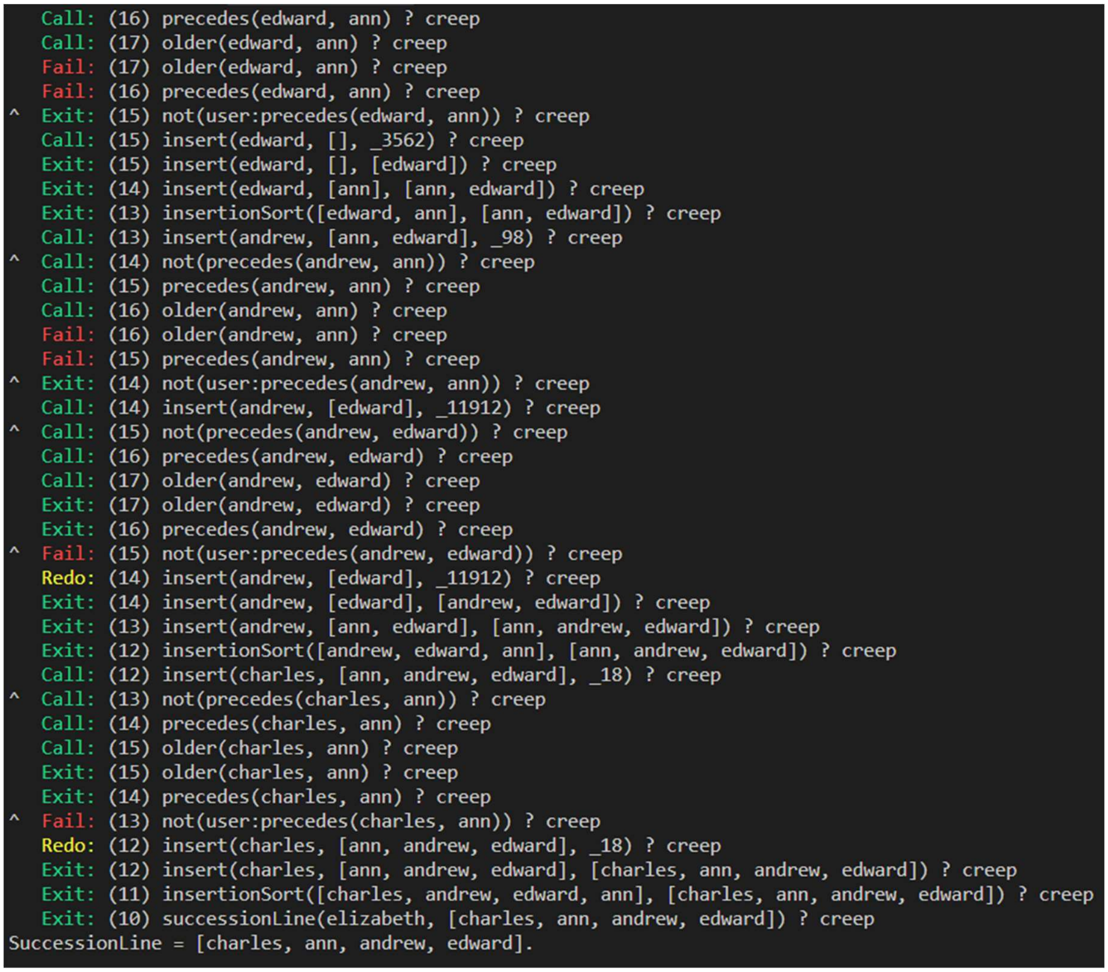


```
%Change from part1 to part2.
%precedes(X,Y) :- prince(X), princess(Y); older(X,Y), prince(X), prince(Y); older(X,Y), princess(X), princess(Y).
precedes(X,Y) :- older(X,Y).
```

We just had to change the rule used by the sorting which is our precedence of X and Y. Since gender is no longer considered, our precedence of variables would just solely be based on age. Hence, we changed our precedence to be reliant on which variable is older.



```
[trace] 2 ?- successionLine(elizabeth, SuccessionLine).
   Call: (10) successionLine(elizabeth, _7140) ? creep
   Call: (11) findAllSuccessors(elizabeth, _8338) ? creep
^  Call: (12) findall(_9094, (successor(_9094, elizabeth), child(_9094, elizabeth)), _8338) ? creep
   Call: (18) successor(_9094, elizabeth) ? creep
   Call: (19) child(_9094, elizabeth) ? creep
   Exit: (19) child(charles, elizabeth) ? creep
   Exit: (18) successor(charles, elizabeth) ? creep
   Call: (18) child(charles, elizabeth) ? creep
   Exit: (18) child(charles, elizabeth) ? creep
   Redo: (19) child(_9094, elizabeth) ? creep
   Exit: (19) child(andrew, elizabeth) ? creep
   Exit: (18) successor(andrew, elizabeth) ? creep
   Call: (18) child(andrew, elizabeth) ? creep
   Exit: (18) child(andrew, elizabeth) ? creep
   Redo: (19) child(_9094, elizabeth) ? creep
   Exit: (19) child(edward, elizabeth) ? creep
   Exit: (18) successor(edward, elizabeth) ? creep
   Call: (18) child(edward, elizabeth) ? creep
   Exit: (18) child(edward, elizabeth) ? creep
   Redo: (19) child(_9094, elizabeth) ? creep
   Exit: (19) child(ann, elizabeth) ? creep
   Exit: (18) successor(ann, elizabeth) ? creep
   Call: (18) child(ann, elizabeth) ? creep
   Exit: (18) child(ann, elizabeth) ? creep
^  Exit: (12) findall(_9094, user:(successor(_9094, elizabeth), child(_9094, elizabeth)), [charles, andrew, edward, ann]) ? creep
   Exit: (11) findAllSuccessors(elizabeth, [charles, andrew, edward, ann]) ? creep
   Call: (11) insertionSort([charles, andrew, edward, ann], _7140) ? creep
   Call: (12) insertionSort([andrew, edward, ann], _28054) ? creep
   Call: (13) insertionSort([edward, ann], _28810) ? creep
   Call: (14) insertionSort([ann], _29566) ? creep
   Call: (15) insertionSort([], _30322) ? creep
   Exit: (15) insertionSort([], []) ? creep
   Call: (15) insert(ann, [], _102) ? creep
   Exit: (15) insert(ann, [], [ann]) ? creep
   Exit: (14) insertionSort([ann], [ann]) ? creep
   Call: (14) insert(edward, [ann], _100) ? creep
^  Call: (15) not(precedes(edward, ann)) ? creep
```



```
   Call: (16) precedes(edward, ann) ? creep
   Call: (17) older(edward, ann) ? creep
   Fail: (17) older(edward, ann) ? creep
   Fail: (16) precedes(edward, ann) ? creep
^  Exit: (15) not(user:precedes(edward, ann)) ? creep
   Call: (15) insert(edward, [], _3562) ? creep
   Exit: (15) insert(edward, [], [edward]) ? creep
   Exit: (14) insert(edward, [ann], [ann, edward]) ? creep
   Exit: (13) insertionSort([edward, ann], [ann, edward]) ? creep
   Call: (13) insert(andrew, [ann, edward], _98) ? creep
^  Call: (14) not(precedes(andrew, ann)) ? creep
   Call: (15) precedes(andrew, ann) ? creep
   Call: (16) older(andrew, ann) ? creep
   Fail: (16) older(andrew, ann) ? creep
   Fail: (15) precedes(andrew, ann) ? creep
^  Exit: (14) not(user:precedes(andrew, ann)) ? creep
   Call: (14) insert(andrew, [edward], _11912) ? creep
^  Call: (15) not(precedes(andrew, edward)) ? creep
   Call: (16) precedes(andrew, edward) ? creep
   Call: (17) older(andrew, edward) ? creep
   Exit: (17) older(andrew, edward) ? creep
   Exit: (16) precedes(andrew, edward) ? creep
^  Fail: (15) not(user:precedes(andrew, edward)) ? creep
   Redo: (14) insert(andrew, [edward], _11912) ? creep
   Exit: (14) insert(andrew, [edward], [andrew, edward]) ? creep
   Exit: (13) insert(andrew, [ann, edward], [ann, andrew, edward]) ? creep
   Exit: (12) insertionSort([andrew, edward, ann], [ann, andrew, edward]) ? creep
   Call: (12) insert(charles, [ann, andrew, edward], _18) ? creep
^  Call: (13) not(precedes(charles, ann)) ? creep
   Call: (14) precedes(charles, ann) ? creep
   Call: (15) older(charles, ann) ? creep
   Exit: (15) older(charles, ann) ? creep
   Exit: (14) precedes(charles, ann) ? creep
^  Fail: (13) not(user:precedes(charles, ann)) ? creep
   Redo: (12) insert(charles, [ann, andrew, edward], _18) ? creep
   Exit: (12) insert(charles, [ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
   Exit: (11) insertionSort([charles, andrew, edward, ann], [charles, ann, andrew, edward]) ? creep
   Exit: (10) successionLine(elizabeth, [charles, ann, andrew, edward]) ? creep
SuccessionLine = [charles, ann, andrew, edward].
```

We decided to use insertion sort as the trace is shorter. However, we did also try quick sort on both Qn 1 and Qn 2. The trace is stored in a text file due to its length however, the results are consistent with the ones we got above.