NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

CZ3005 Artificial Intelligence - Assignment 3

Learning to Use Prolog as a Logic Programming Tool

Bisakha Das
DSAI
U1823460E

Table of Contents

Exercise 1: The Smart Phone Rivalry

1.1 Translation and First Order Logic (FOL)

Text:
sumsum, a competitor of appy, developed some nice smart phone technology called galactica- s3, all of which was stolen by stevey, who is a boss. It is unethical for a boss to steal business from rival companies. A competitor of appy is a rival. Smart phone technology is business.

The above text has been broken down into its constituent simple and compound sentences for ease of translation to First Order Logic (FOL). The table below shows each of those sentences beside the translated FOL. Additional translation has been provided for some sentences for ease of understanding the FOL.

	Sentence	**First Order Logic**
1	Sumsum is a company	company(sumSum).
2	Appy is a company	company(appy).
3	Galactica-S3 is a smart phone technology	smartPhoneTech(galacticaS3).
4	Stevey is a boss	boss(stevey).
5	Sumsum is a competitor of appy	competitor(sumsum,appy).
6	Sumsum developed galactica- s3	developedBy(sumsum, galacticaS3).
7	Stevey stole galactica- s3	stole(stevey,galacticaS3).
8	A competitor of appy is a rival.	∀ X, competitor(X, appy) ∨ competitor(appy, X) ⇒ rival(X)
9	Smart phone technology is business. Translation: For all X which is a smart phone technology, it is also a business	∀ X, smartPhoneTech(X) ⇒ business(X)
10	It is unethical for a boss to steal business from rival companies. Translation: If a boss X steals a business Y developed by a rival company Z, it is unethical.	∀ X, Y, Z, boss(X) ∧ stole(X, Y) ∧ business(Y) ∧ developedBy(Z, Y) ∧ rival(Z) ⇒ unethical(X)

1.2 Prolog Statements

The FOL statements from the previous parts have been translated into Prolog clauses in the table below:

	First Order Logic	**Prolog**
1	company(sumSum).	company(sumSum).
2	company(appy).	company(appy).
3	smartPhoneTech(galacticaS3).	smartPhoneTech(galacticaS3).
4	boss(stevey).	boss(stevey).
5	competitor(sumsum,appy).	competitor(sumsum,appy).
6	developedBy(sumsum, galacticaS3).	developedBy(sumsum, galacticaS3).
7	stole(stevey,galacticaS3).	stole(stevey,galacticaS3).
8	∀ X, competitor(X, appy) ∨ competitor(appy, X) ⇒ rival(X)	rival(X) :- competitor(X,appy);competitor(appy,X).
9	∀ X, smartPhoneTech(X) ⇒ business(X)	business(X) :- smartPhoneTech(X).
10	∀ X, Y, Z, boss(X) ∧ stole(X, Y) ∧ business(Y) ∧ developedBy(Z, Y) ∧ rival(Z) ⇒ unethical(X)	unethical(X):- boss(X), stole(X,Y), business(Y), developedBy(Z,Y), rival(Z).

1.2.1 Final Prolog clauses

```
company(sumSum).
company(appy).
smartPhoneTech(galacticaS3).
boss(stevey).
competitor(sumsum,appy).
develop(sumsum, galacticaS3).
stole(stevey,galacticaS3).
rival(X) :- competitor(X,appy);competitor(appy,X).
business(X) :- smartPhoneTech(X).
unethical(X):- boss(X), stole(X,Y), business(Y), develop(Z,Y), rival(Z).
```

1.3 Proof and Trace

```
% /Users/dasbisakha/Desktop/NTU/Artificial
Intelligence/Assignment3/BisakhaDas_qn_1_2.pl compiled 0.00 sec, 10 clauses
?- unethical(stevey).
true .

?- trace.
true.

[trace] ?- unethical(stevey).
Call: (10) unethical(stevey) ? creep
Call: (11) boss(stevey) ? creep
Exit: (11) boss(stevey) ? creep
Call: (11) stole(stevey, _658) ? creep
Exit: (11) stole(stevey, galacticaS3) ? creep
Call: (11) business(galacticaS3) ? creep
Call: (12) smartPhoneTech(galacticaS3) ? creep
Exit: (12) smartPhoneTech(galacticaS3) ? creep
Exit: (11) business(galacticaS3) ? creep
Call: (11) develop(_920, galacticaS3) ? creep
Exit: (11) develop(sumsum, galacticaS3) ? creep
Call: (11) rival(sumsum) ? creep
Call: (12) competitor(sumsum, appy) ? creep
Exit: (12) competitor(sumsum, appy) ? creep
Exit: (11) rival(sumsum) ? creep
Exit: (10) unethical(stevey) ? creep
true .
```

- End of Question 1 -

Exercise 2: The Royal Family

2.1 Old Succession Rule

Text:
The old Royal succession rule states that the throne is passed down along the male line according to the order of birth before the consideration along the female line – similarly according to the order of birth. queen elizabeth, the monarch of United Kingdom, has four offsprings; namely:- prince charles, princess ann, prince andrew and prince edward – listed in the order of birth.

Given the old succession rules, the succession line is prince charles, prince andrew, prince edward and princess ann.

2.1.1 Prolog

The above text has been broken down into its constituent simple and compound sentences for ease of translation to Prolog clauses. The table below shows each of those sentences beside the translated Prolog clauses. Additional to defining facts and relations, information which do not directly depend on the text above are also included in the table, such as bubble sorting.

	Sentence/ Rules	**Prolog Clause**
1	queen Elizabeth has four offsprings; namely:- prince charles, princess ann, prince andrew and prince edward	offspring(prince, charles).
2		offspring(princess, ann).
3		offspring(prince, andrew).
4		offspring(prince, edward).
5	listed in the order of birth.	older(charles, ann).
6		older(ann, andrew).
7		older(andrew, edward).
8	male line	male(A):- offspring(prince,A).
9	female line	female(A):- offspring(princess,A).
10	Defining rules such that if X is older than Y, and Y is older than Z, then X is older than Z. $\forall x,y,z \ \ is\_older(x,y) \ \land is\_older(y,z) \Rightarrow is\_older(x,z)$	is_older(X, Y):- older(X, Y).
11		is_older(A, B):- older(A, X),is_older(X, B).
12	Defining the order of succession: First priority is given to gender, followed by age. Translation: If one person is a prince and one a princess, then prince gets first priority. If the two people are the same gender, then the older gets higher preference.	in_order(X, Y) :- offspring(prince, X), offspring(princess, Y).
13		in_order(X, Y) :- offspring(A, X), offspring(A, Y), is_older(X, Y).
14	Performing bubble sort using an accumulator	bubble_sort(List,Sorted):- b_sort(List,[],Sorted).

15		b_sort([],Acc,Acc).
16		b_sort([H|T],Acc,Sorted):-bubble(H,T,NT,Max),b_sort(NT,[Max|Acc],Sorted).
17		bubble(X,[Y|T],[X|NT],Max):-in_order(X, Y),bubble(Y,T,NT,Max).
18		bubble(X,[Y|T],[Y|NT],Max):-not(in_order(X, Y)),bubble(X,T,NT,Max).
19		bubble(X,[],[],X).
20	Getting the final succession line	successionLine(X):-findall(Y,offspring(_,Y),OffspringList),bubble_sort(OffspringList,X).

2.1.2 Final Prolog clauses

```
offspring(prince, charles).
offspring(princess, ann).
offspring(prince, andrew).
offspring(prince, edward).
older(charles, ann).
older(ann, andrew).
older(andrew, edward).
male(A):- offspring(prince,A).
female(A):- offspring(princess,A).
is_older(X, Y):- older(X, Y).
is_older(A, B):- older(A, X),is_older(X, B).
in_order(X, Y) :- offspring(prince, X), offspring(princess, Y).
in_order(X, Y) :- offspring(A, X), offspring(A, Y), is_older(X, Y).
bubble_sort(List,Sorted):-b_sort(List,[],Sorted).
b_sort([],Acc,Acc).
b_sort([H|T],Acc,Sorted):-bubble(H,T,NT,Max),b_sort(NT,[Max|Acc],Sorted).
bubble(X,[Y|T],[X|NT],Max):-in_order(X, Y),bubble(Y,T,NT,Max).
bubble(X,[Y|T],[Y|NT],Max):- not(in_order(X, Y)),bubble(X,T,NT,Max).
bubble(X,[],[],X).
successionLine(X):-
findall(Y,offspring(_,Y),OffspringList),bubble_sort(OffspringList,X).
```

2.1.3 Trace and Results

```
% /Users/dasbisakha/Desktop/NTU/Artificial Intelligence/Assignment3/BisakhaDas_qn_2_1.pl
compiled 0.00 sec, 20 clauses
?- successionLine(X).
X = [charles, andrew, edward, ann] .

?- trace.
true.

[trace] ?- successionLine(X).
Call: (10) successionLine(_14380) ? creep
^ Call: (11) findall(_14770, offspring(_14768, _14770), _14830) ? creep
Call: (16) offspring(_14768, _14770) ? creep
Exit: (16) offspring(prince, charles) ? creep
Redo: (16) offspring(_14768, _14770) ? creep
Exit: (16) offspring(princess, ann) ? creep
Redo: (16) offspring(_14768, _14770) ? creep
Exit: (16) offspring(prince, andrew) ? creep
Redo: (16) offspring(_14768, _14770) ? creep
Exit: (16) offspring(prince, edward) ? creep
^ Exit: (11) findall(_14770, user:offspring(_14768, _14770), [charles, ann, andrew, edward]) ? creep
Call: (11) bubble_sort([charles, ann, andrew, edward], _14380) ? creep
Call: (12) b_sort([charles, ann, andrew, edward], [], _14380) ? creep
Call: (13) bubble(charles, [ann, andrew, edward], _15416, _15418) ? creep
Call: (14) in_order(charles, ann) ? creep
Call: (15) offspring(prince, charles) ? creep
Exit: (15) offspring(prince, charles) ? creep
Call: (15) offspring(princess, ann) ? creep
Exit: (15) offspring(princess, ann) ? creep
Exit: (14) in_order(charles, ann) ? creep
Call: (14) bubble(ann, [andrew, edward], _15406, _15732) ? creep
Call: (15) in_order(ann, andrew) ? creep
Call: (16) offspring(prince, ann) ? creep
Fail: (16) offspring(prince, ann) ? creep
Redo: (15) in_order(ann, andrew) ? creep
Call: (16) offspring(_15952, ann) ? creep
Exit: (16) offspring(princess, ann) ? creep
Call: (16) offspring(princess, andrew) ? creep
Fail: (16) offspring(princess, andrew) ? creep
Fail: (15) in_order(ann, andrew) ? creep
Redo: (14) bubble(ann, [andrew, edward], _15406, _16178) ? creep
^ Call: (15) not(in_order(ann, andrew)) ? creep
Call: (16) in_order(ann, andrew) ? creep
Call: (17) offspring(prince, ann) ? creep
Fail: (17) offspring(prince, ann) ? creep
Redo: (16) in_order(ann, andrew) ? creep
Call: (17) offspring(_16454, ann) ? creep
Exit: (17) offspring(princess, ann) ? creep
Call: (17) offspring(princess, andrew) ? creep
```

```
Fail: (17) offspring(princess, andrew) ? creep
Fail: (16) in_order(ann, andrew) ? creep
^ Exit: (15) not(user:in_order(ann, andrew)) ? creep
Call: (15) bubble(ann, [edward], _16166, _16724) ? creep
Call: (16) in_order(ann, edward) ? creep
Call: (17) offspring(prince, ann) ? creep
Fail: (17) offspring(prince, ann) ? creep
Redo: (16) in_order(ann, edward) ? creep
Call: (17) offspring(_16944, ann) ? creep
Exit: (17) offspring(princess, ann) ? creep
Call: (17) offspring(princess, edward) ? creep
Fail: (17) offspring(princess, edward) ? creep
Fail: (16) in_order(ann, edward) ? creep
Redo: (15) bubble(ann, [edward], _16166, _17170) ? creep
^ Call: (16) not(in_order(ann, edward)) ? creep
Call: (17) in_order(ann, edward) ? creep
Call: (18) offspring(prince, ann) ? creep
Fail: (18) offspring(prince, ann) ? creep
Redo: (17) in_order(ann, edward) ? creep
Call: (18) offspring(_17446, ann) ? creep
Exit: (18) offspring(princess, ann) ? creep
Call: (18) offspring(princess, edward)creep
Fail: (18) offspring(princess, edward) ? ? creep
Fail: (17) in_order(ann, edward) ? creep
^ Exit: (16) not(user:in_order(ann, edward)) ? creep
Call: (16) bubble(ann, [], _17158, _17716)creep
Exit: (16) bubble(ann, [], [], ann) ? ? creep
Exit: (15) bubble(ann, [edward], [edward], ann) ? creep
Exit: (14) bubble(ann, [andrew, edward], [andrew, edward], ann) ? creep
Exit: (13) bubble(charles, [ann, andrew, edward], [charles, andrew, edward], ann) ? creep
Call: (13) b_sort([charles, andrew, edward], [ann], _14380) ? creep
Call: (14) bubble(charles, [andrew, edward], _17984, _17986) ? creep
Call: (15) in_order(charles, andrew) ? creep
Call: (16) offspring(prince, charles) ? creep
Exit: (16) offspring(prince, charles) ? creep
Call: (16) offspring(princess, andrew)creep
Fail: (16) offspring(princess, andrew) ? ? creep
Redo: (15) in_order(charles, andrew) ? creep
Call: (16) offspring(_18294, charles) ? creep
Exit: (16) offspring(prince, charles) ? creep
Call: (16) offspring(prince, andrew) ? creep
Exit: (16) offspring(prince, andrew) ? creep
Call: (16) is_older(charles, andrew) ? creep
Call: (17) older(charles, andrew) ? creep
Fail: (17) older(charles, andrew) ? creep
Redo: (16) is_older(charles, andrew)creep
Call: (17) older(charles, _18648) ? ? creep
Exit: (17) older(charles, ann) ? creep
Call: (17) is_older(ann, andrew) ? creep
Call: (18) older(ann, andrew) ? creep
```

```
Exit: (18) older(ann, andrew) ? creep
Exit: (17) is_older(ann, andrew) ? creep
Exit: (16) is_older(charles, andrew) ? creep
Exit: (15) in_order(charles, andrew) ? creep
Call: (15) bubble(andrew, [edward], _17974, _19004) ? creep
Call: (16) in_order(andrew, edward) ? creep
Call: (17) offspring(prince, andrew)creep
Exit: (17) offspring(prince, andrew) ? ? creep
Call: (17) offspring(princess, edward) ? creep
Fail: (17) offspring(princess, edward) ? creep
Redo: (16) in_order(andrew, edward) ? creep
Call: (17) offspring(_19312, andrew) ? creep
Exit: (17) offspring(prince, andrew) ? creep
Call: (17) offspring(prince, edward) ? creep
Exit: (17) offspring(prince, edward) ? creep
Call: (17) is_older(andrew, edward) ? creep
Call: (18) older(andrew, edward) ? creep
Exit: (18) older(andrew, edward) ? creep
Exit: (17) is_older(andrew, edward) ? creep
Exit: (16) in_order(andrew, edward) ? creep
Call: (16) bubble(edward, [], _18992, _19714) ? creep
Exit: (16) bubble(edward, [], [], edward) ? creep
Exit: (15) bubble(andrew, [edward], [andrew], edward) ? creep
Exit: (14) bubble(charles, [andrew, edward], [charles, andrew], edward) ? creep
Call: (14) b_sort([charles, andrew], [edward, ann], _14380) ? creep
Call: (15) bubble(charles, [andrew], _19938, _19940) ? creep
Call: (16) in_order(charles, andrew) ? creep
Call: (17) offspring(prince, charles) ? creep
Exit: (17) offspring(prince, charles) ? creep
Call: (17) offspring(princess, andrew) ? creep
Fail: (17) offspring(princess, andrew) ? creep
Redo: (16) in_order(charles, andrew) ? creep
Call: (17) offspring(_20248, charles) ? creep
Exit: (17) offspring(prince, charles) ? creep
Call: (17) offspring(prince, andrew) ? creep
Exit: (17) offspring(prince, andrew) ? creep
Call: (17) is_older(charles, andrew) ? creep
Call: (18) older(charles, andrew) ? creep
Fail: (18) older(charles, andrew) ? creep
Redo: (17) is_older(charles, andrew) ? creep
Call: (18) older(charles, _20602) ? creep
Exit: (18) older(charles, ann) ? creep
Call: (18) is_older(ann, andrew) ? creep
Call: (19) older(ann, andrew) ? creep
Exit: (19) older(ann, andrew) ? creep
Exit: (18) is_older(ann, andrew) ? creep
Exit: (17) is_older(charles, andrew) ? creep
Exit: (16) in_order(charles, andrew) ? creep
Call: (16) bubble(andrew, [], _19928, _20958) ? creep
Exit: (16) bubble(andrew, [], [], andrew)creep
```

```
Exit: (15) bubble(charles, [andrew], [charles], andrew) ? ? creep
Call: (15) b_sort([charles], [andrew, edward, ann], _14380) ? creep
Call: (16) bubble(charles, [], _21138, _21140) ? creep
Exit: (16) bubble(charles, [], [], charles) ? creep
Call: (16) b_sort([], [charles, andrew, edward, ann], _14380) ? creep
Exit: (16) b_sort([], [charles, andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
Exit: (15) b_sort([charles], [andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
Exit: (14) b_sort([charles, andrew], [edward, ann], [charles, andrew, edward, ann]) ? creep
Exit: (13) b_sort([charles, andrew, edward], [ann], [charles, andrew, edward, ann]) ? creep
Exit: (12) b_sort([charles, ann, andrew, edward], [], [charles, andrew, edward, ann]) ? creep
Exit: (11) bubble_sort([charles, ann, andrew, edward], [charles, andrew, edward, ann]) ? creep
Exit: (10) successionLine([charles, andrew, edward, ann]) ? creep
X = [charles, andrew, edward, ann] .
```

2.2 New Succession Rule

Given the new succession rules, where gender is not taken into consideration while determining the successor, the succession line is prince charles, princess ann, prince andrew and prince edward.

2.2.1 Changes required from the Old Succession Rule

The table below shows the Prolog clauses of the old succession rule along with reasonings behind required changes to convert it to suit the new succession rules.

	Old Succession Rule Prolog Clause	**Changes Required and reason**
1	offspring(prince, charles).	These clauses are still required to declare facts such as the names of the offsprings and who is older among a pair of them.
2	offspring(princess, ann).	
3	offspring(prince, andrew).	
4	offspring(prince, edward).	
5	older(charles, ann).	
6	older(ann, andrew).	
7	older(andrew, edward).	
8	male(A):- offspring(prince,A).	These clauses can be removed, since there is no more a need to differentiate between males and females.
9	female(A):- offspring(princess,A).	
10	is_older(X, Y):- older(X, Y).	These clauses are still required for defining rules such that if X is older than Y, and Y is older than Z, then X is older than Z. $\forall x, y, z \;\; is\_older(x, y) \; \land is\_older(y, z) \Rightarrow is\_older(x, z)$
11	is_older(A, B):- older(A, X),is_older(X, B).	
12	in_order(X, Y) :- offspring(prince, X), offspring(princess, Y).	These clauses are no longer necessary, since these primarily helped differentiate between princes and princesses. We can

13	in_order(X, Y) :- offspring(A, X), offspring(A, Y), is_older(X, Y).	later define the parameters for bubble sort to be the is_older clause instead of this in_order clause, because without differentiation between genders, these clauses essentially mean the same thing as the is_older clause.
14	bubble_sort(List,Sorted):- b_sort(List,[],Sorted).	The bubble sort is still necessary, but since the in_order clause is no longer necessary (explained in the above row), it needs to be changed to is_older here. bubble_sort(List,Sorted):- b_sort(List,[],Sorted). b_sort([],Acc,Acc). b_sort([H|T],Acc,Sorted):- bubble(H,T,NT,Max),b_sort(NT,[Max|Acc] ,Sorted). bubble(X,[Y|T],[X|NT],Max):-is_older(X, Y),bubble(Y,T,NT,Max). bubble(X,[Y|T],[Y|NT],Max):- not(is_older(X, Y)),bubble(X,T,NT,Max). bubble(X,[],[],X).
15	b_sort([],Acc,Acc).	
16	b_sort([H|T],Acc,Sorted):- bubble(H,T,NT,Max),b_sort(NT,[Max|Acc] ,Sorted).	
17	bubble(X,[Y|T],[X|NT],Max):-in_order(X, Y),bubble(Y,T,NT,Max).	
18	bubble(X,[Y|T],[Y|NT],Max):- not(in_order(X, Y)),bubble(X,T,NT,Max).	
19	bubble(X,[],[],X).	
20	successionLine(X):- findall(Y,offspring(_,Y),OffspringList),bubble_sort(OffspringList,X).	There is no change required for this clause

2.2.2 Final Prolog clauses for new succession rule

```
offspring(prince, charles).
offspring(princess, ann).
offspring(prince, andrew).
offspring(prince, edward).
older(charles, ann).
older(ann, andrew).
older(andrew, edward).
is_older(X, Y):- older(X, Y).
is_older(A, B):- older(A, X),is_older(X, B).
bubble_sort(List,Sorted):-b_sort(List,[],Sorted).
b_sort([],Acc,Acc).
b_sort([H|T],Acc,Sorted):-bubble(H,T,NT,Max),b_sort(NT,[Max|Acc],Sorted).
bubble(X,[Y|T],[X|NT],Max):-is_older(X, Y),bubble(Y,T,NT,Max).
bubble(X,[Y|T],[Y|NT],Max):- not(is_older(X, Y)),bubble(X,T,NT,Max).
bubble(X,[],[],X).
successionLine(X):-
findall(Y,offspring(_,Y),OffspringList),bubble_sort(OffspringList,X).
```

2.2.3 Trace and Results

```
% /Users/dasbisakha/Desktop/NTU/Artificial Intelligence/Assignment3/BisakhaDas_qn_2_2.pl
compiled 0.00 sec, 16 clauses
?- successionLine(X).
X = [charles, ann, andrew, edward].

?- trace.
true.

[trace] ?- successionLine(X).
Call: (10) successionLine(_14350) ? creep
^ Call: (11) findall(_14740, offspring(_14738, _14740), _14800) ? creep
Call: (16) offspring(_14738, _14740) ? creep
Exit: (16) offspring(prince, charles) ? creep
Redo: (16) offspring(_14738, _14740) ? creep
Exit: (16) offspring(princess, ann) ? creep
Redo: (16) offspring(_14738, _14740) ? creep
Exit: (16) offspring(prince, andrew) ? creep
Redo: (16) offspring(_14738, _14740) ? creep
Exit: (16) offspring(prince, edward) ? creep
^ Exit: (11) findall(_14740, user:offspring(_14738, _14740), [charles, ann, andrew, edward]) ?
creep
Call: (11) bubble_sort([charles, ann, andrew, edward], _14350) ? creep
Call: (12) b_sort([charles, ann, andrew, edward], [], _14350) ? creep
Call: (13) bubble(charles, [ann, andrew, edward], _15386, _15388) ? creep
Call: (14) is_older(charles, ann) ? creep
Call: (15) older(charles, ann) ? creep
Exit: (15) older(charles, ann) ? creep
Exit: (14) is_older(charles, ann) ? creep
Call: (14) bubble(ann, [andrew, edward], _15376, _15614) ? creep
Call: (15) is_older(ann, andrew) ? creep
Call: (16) older(ann, andrew) ? creep
Exit: (16) older(ann, andrew) ? creep
Exit: (15) is_older(ann, andrew) ? creep
Call: (15) bubble(andrew, [edward], _15602, _15840) ? creep
Call: (16) is_older(andrew, edward) ? creep
Call: (17) older(andrew, edward) ? creep
Exit: (17) older(andrew, edward) ? creep
Exit: (16) is_older(andrew, edward) ? creep
Call: (16) bubble(edward, [], _15828, _16066) ? creep
Exit: (16) bubble(edward, [], [], edward) ? creep
Exit: (15) bubble(andrew, [edward], [andrew], edward) ? creep
Exit: (14) bubble(ann, [andrew, edward], [ann, andrew], edward) ? creep
Exit: (13) bubble(charles, [ann, andrew, edward], [charles, ann, andrew], edward) ? creep
Call: (13) b_sort([charles, ann, andrew], [edward], _14350) ? creep
Call: (14) bubble(charles, [ann, andrew], _16334, _16336) ? creep
Call: (15) is_older(charles, ann) ? creep
Call: (16) older(charles, ann) ? creep
Exit: (16) older(charles, ann) ? creep
```

```
Exit: (15) is_older(charles, ann) ? creep
Call: (15) bubble(ann, [andrew], _16324, _16562) ? creep
Call: (16) is_older(ann, andrew) ? creep
Call: (17) older(ann, andrew) ? creep
Exit: (17) older(ann, andrew) ? creep
Exit: (16) is_older(ann, andrew) ? creep
Call: (16) bubble(andrew, [], _16550, _16788) ? creep
Exit: (16) bubble(andrew, [], [], andrew) ? creep
Exit: (15) bubble(ann, [andrew], [ann], andrew) ? creep
Exit: (14) bubble(charles, [ann, andrew], [charles, ann], andrew) ? creep
Call: (14) b_sort([charles, ann], [andrew, edward], _14350) ? creep
Call: (15) bubble(charles, [ann], _17012, _17014) ? creep
Call: (16) is_older(charles, ann) ? creep
Call: (17) older(charles, ann) ? creep
Exit: (17) older(charles, ann) ? creep
Exit: (16) is_older(charles, ann) ? creep
Call: (16) bubble(ann, [], _17002, _17240) ? creep
Exit: (16) bubble(ann, [], [], ann) ? creep
Exit: (15) bubble(charles, [ann], [charles], ann) ? creep
Call: (15) b_sort([charles], [ann, andrew, edward], _14350) ? creep
Call: (16) bubble(charles, [], _17420, _17422) ? creep
Exit: (16) bubble(charles, [], [], charles) ? creep
Call: (16) b_sort([], [charles, ann, andrew, edward], _14350) ? creep
Exit: (16) b_sort([], [charles, ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
Exit: (15) b_sort([charles], [ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
Exit: (14) b_sort([charles, ann], [andrew, edward], [charles, ann, andrew, edward]) ? creep
Exit: (13) b_sort([charles, ann, andrew], [edward], [charles, ann, andrew, edward]) ? creep
Exit: (12) b_sort([charles, ann, andrew, edward], [], [charles, ann, andrew, edward]) ? creep
Exit: (11) bubble_sort([charles, ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
Exit: (10) successionLine([charles, ann, andrew, edward]) ? creep
X = [charles, ann, andrew, edward] .
```

- End of Question 2 –

- End of Assignment 3 -