NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC3000/CZ3005: Artificial Intelligence

Assignment 2

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Name	Matric No.	Contribution
Tran Que An	U2120901G	Worked on Exercise 1
Lee Wen Wei	U2120178J	Ensure correctness of both exercise and collaborated with both members.
Sandhiya Sukumaran	U2120682B	Worked on Exercise 2

Exercise 1:

1. Translate the natural language statements above describing the dealing within the Smart Phone industry into First Order Logic (FOL).

In this section, we will capitalize sumsum, appy, and functions as normal FOL convention.

Based on common sense, we can assume that “sumsum” and “appy” are companies. Hence, we have these following FOL statements:

Company(SumSum)
Company(Appy)

“sumsum, a competitor of appy” means that SumSum is a competitor of Appy

Competitor(SumSum,Appy)

Based on common sense, the fact that SumSum is a competitor of Appy also means that Appy is a competitor of SumSum

Competitor(Appy, SumSum)

“developed some nice smart phone technology called galactica- s3”

Developed(SumSum,GalacticaS3)

“all of which was stolen by stevey, who is a boss of appy”

Stolen(Stevey, GalacticaS3)
Boss(Stevey, Appy)

“A competitor of is a rival.”

$\forall$ x, y Competitor(x,y) $\Rightarrow$ Rival(x,y)

“Smart phone technology is business.”

Technology(Galacticas3)
$\forall$ x (Technology(x) $\Rightarrow$ Business(x))

“It is unethical for a boss to steal business from rival companies” means that a boss X is unethical if and only if:

x is the boss of a company y: Boss(x,y) $\wedge$ Company(y)
x stole a business z: Stolen(x,z) $\wedge$ Business(z)
z was developed by another company t: Developed(t,z) $\wedge$ Company(t)
and t is the rival company of y: Rival(y,t)

Hence,

$\exists$ x, y, z, t Unethical(x) $\Leftrightarrow$ Boss(x,y) $\wedge$ Company(y) $\wedge$ Stolen(x,z) $\wedge$ Business(z) $\wedge$ Developed(t,z) $\wedge$ Company(t) $\wedge$ Rival(y,t)

In summary, here are our FOL statements:

Company(SumSum)

Company(Appy)
Competitor(SumSum,Appy)
Competitor(Appy, SumSum)
Developed(SumSum,GalacticaS3)
Stolen(Stevey, GalacticaS3)
Boss(Stevey, Appy)
Competitor(x,y) ⇒ Rival(x,y)
Technology(Galacticas3)
Technology(x) ⇒ Business(x)
Unethical(x) ⇔ Boss(x,y) ∧ Company(y) ∧ Stolen(x,z) ∧ Business(z) ∧ Developed(t,z) ∧ Company(t) ∧ Rival(y,t)

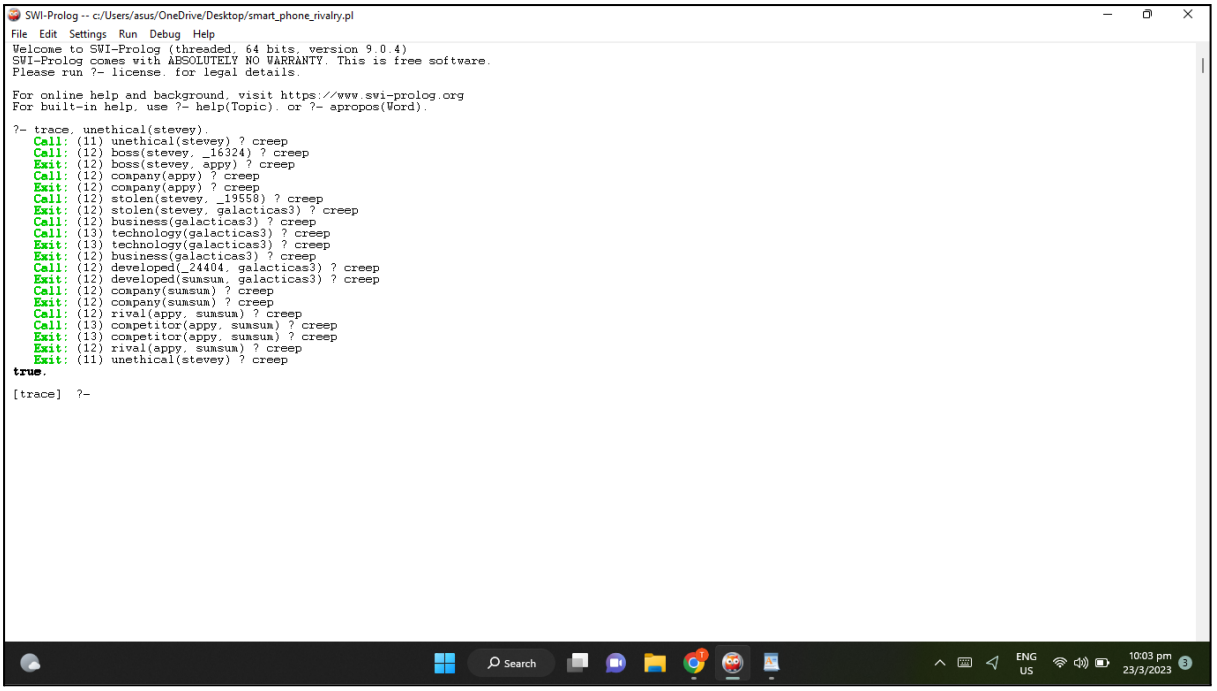
2. Write these FOL statements as Prolog clauses.

We can write above FOL statements as Prolog clauses, as follows:

```
company(sumsum).
company(appy).

competitor(sumsum,appy).
competitor(appy,sumsum).

developed(sumsum,galacticas3).

boss(stevey,appy).
stolen(stevey,galacticas3).

technology(galacticas3).
business(X) :- technology(X).

rival(X,Y) :- competitor(X,Y).

unethical(X) :- boss(X,Y), company(Y), stolen(X,Z), business(Z), developed(T,Z), company(T), rival(Y,T).
```

3. Using Prolog, prove that Stevey is unethical. Show a trace of your proof.

Query “trace, unethical(stevey)” from these above Prolog clauses, we have these results:



The result is “**true**”, hence, Stevey is unethical.

Exercise 2: The Royal Family

1. Define their relations and rules in a Prolog rule base. Hence, define the old Royal succession rule. Using this old succession rule determine the line of succession based on the information given. Do a trace to show your results.

Relations and rules:

```
queen(queen_elizabeth).

declaring the females:
female(queen_elizabeth).
female(princess_ann).

declaring the males:
male(prince_charles).
male(prince_andrew).
male(prince_edward).

declaring the offsprings of queen elizabeth:
offspring(prince_charles, queen_elizabeth).
offspring(princess_ann, queen_elizabeth).
offspring(prince_andrew, queen_elizabeth).
offspring(prince_edward, queen_elizabeth).

declaring who is older:
older_than(prince_charles, princess_ann).
older_than(princess_ann, prince_andrew).
older_than(prince_andrew, prince_edward).
older(A,B):- older_than(A,B).
older(A,B):- older_than(A,X), older(X,B).

to check successor:
successor(X, Y):- offspring(Y, X)

to list out all successors of X:
successionList(X, SuccessionList):-
        findall(Y, successor(X, Y), SuccessionList).
```

** For rules of the succession ordering, they must be offspring of the same parent and must not be the existing queen

(1) Male child and female child so return male
precedes(X, Y):- offspring(X, A), offspring(Y, A), male(X), female(Y), not(queen(Y)).

(2) Both male child so return the older male
precedes(X, Y):- offspring(X, A), offspring(Y, A), male(X), male(Y), older(X, Y).

(3) Both female child so return the older female
precedes(X, Y):- offspring(X, A), offspring(Y, A), female(X), female(Y), older(X, Y), not(queen(X)), not(queen(Y)).

Algorithm to sort the succession list:
sort_succession_list([A|B], SortedList):- sort_succession_list(B, Sorted_Tail), insert(A, Sorted_Tail, SortedList).
sort_succession_list([], []).
insert(A, [B|C], [B|D]):- not(precedes(A,B)), !, insert(A, C, D).
insert(A, C, [A|C]).

Returning a sorted succession list:
sortedSuccessionList(X, SuccessionList):- findall(Y, offspring(Y,X), Offspring), sort_succession_list(Offspring, SuccessionList).

Trace:

```
   trace, sortedSuccessionList(queen_elizabeth, SuccessionList).
 Call: (11) sortedSuccessionList(queen_elizabeth, _1278) ? creep
 Call: (12) findall(_2688, offspring(_2688, queen_elizabeth), _2696) ? creep
 Call: (17) offspring(_2688, queen_elizabeth) ? creep
 Exit: (17) offspring(prince_charles, queen_elizabeth) ? creep
 Redo: (17) offspring(_2688, queen_elizabeth) ? creep
 Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
 Redo: (17) offspring(_2688, queen_elizabeth) ? creep
 Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
 Redo: (17) offspring(_2688, queen_elizabeth) ? creep
 Exit: (17) offspring(prince_edward, queen_elizabeth) ? creep
 Exit: (12) findall(_2688, user:offspring(_2688, queen_elizabeth), [prince_charles, princess_ann, prince_andrew, prince_edward]) ? creep
 Call: (12) sort_succession_list([prince_charles, princess_ann, prince_andrew, prince_edward], _1278) ? creep
 Call: (13) sort_succession_list([princess_ann, prince_andrew, prince_edward], _11698) ? creep
 Call: (14) sort_succession_list([prince_andrew, prince_edward], _12510) ? creep
 Call: (15) sort_succession_list([prince_edward], _13322) ? creep
 Call: (16) sort_succession_list([], _14134) ? creep
 Exit: (16) sort_succession_list([], []) ? creep
 Call: (16) insert(prince_edward, [], _13322) ? creep
 Exit: (16) insert(prince_edward, [], [prince_edward]) ? creep
 Exit: (15) sort_succession_list([prince_edward], [prince_edward]) ? creep
 Call: (15) insert(prince_andrew, [prince_edward], _12510) ? creep
 Call: (16) not(precedes(prince_andrew, prince_edward)) ? creep
 Call: (17) precedes(prince_andrew, prince_edward) ? creep
 Call: (18) offspring(prince_andrew, _20660) ? creep
 Exit: (18) offspring(prince_andrew, queen_elizabeth) ? creep
 Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
 Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
 Call: (18) male(prince_andrew) ? creep
 Exit: (18) male(prince_andrew) ? creep
 Call: (18) male(prince_edward) ? creep
 Exit: (18) male(prince_edward) ? creep
 Call: (18) older(prince_andrew, prince_edward) ? creep
 Call: (19) older_than(prince_andrew, prince_edward) ? creep
 Exit: (19) older_than(prince_andrew, prince_edward) ? creep
 Exit: (18) older(prince_andrew, prince_edward) ? creep
 Exit: (17) precedes(prince_andrew, prince_edward) ? creep
 Fail: (16) not(user:precedes(prince_andrew, prince_edward)) ? creep
 Redo: (15) insert(prince_andrew, [prince_edward], _112) ? creep
 Exit: (15) insert(prince_andrew, [prince_edward], [prince_andrew, prince_edward]) ? creep
 Exit: (14) sort_succession_list([prince_andrew, prince_edward], [prince_andrew, prince_edward]) ? creep
 Call: (14) insert(princess_ann, [prince_andrew, prince_edward], _110) ? creep
 Call: (15) not(precedes(princess_ann, prince_andrew)) ? creep
 Call: (16) precedes(princess_ann, prince_andrew) ? creep
 Call: (17) offspring(princess_ann, _5622) ? creep
 Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
 Call: (17) offspring(prince_andrew, queen_elizabeth) ? creep
 Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
 Call: (17) male(princess_ann) ? creep
 Fail: (17) male(princess_ann) ? creep
 Redo: (16) precedes(princess_ann, prince_andrew) ? creep
 Call: (17) offspring(princess_ann, _11286) ? creep
 Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
 Call: (17) offspring(prince_andrew, queen_elizabeth) ? creep
 Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
 Call: (17) female(princess_ann) ? creep
 Exit: (17) female(princess_ann) ? creep
 Call: (17) female(prince_andrew) ? creep
 Fail: (17) female(prince_andrew) ? creep
 Redo: (16) precedes(princess_ann, prince_andrew) ? creep
 Call: (17) offspring(princess_ann, _18562) ? creep
 Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
 Call: (17) offspring(prince_andrew, queen_elizabeth) ? creep
 Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
 Call: (17) male(princess_ann) ? creep
 Fail: (17) male(princess_ann) ? creep
 Fail: (16) precedes(princess_ann, prince_andrew) ? creep
 Exit: (15) not(user:precedes(princess_ann, prince_andrew)) ? creep
 Call: (15) insert(princess_ann, [prince_edward], _3980) ? creep
 Call: (16) not(precedes(princess_ann, prince_edward)) ? creep
 Call: (17) precedes(princess_ann, prince_edward) ? creep
 Call: (18) offspring(princess_ann, _27504) ? creep
 Exit: (18) offspring(princess_ann, queen_elizabeth) ? creep
 Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
 Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
 Call: (18) male(princess_ann) ?
```

```
   Fail: (17) male(princess_ann) ? creep
   Fail: (16) precedes(princess_ann, prince_andrew) ? creep
 ^ Exit: (15) not(user:precedes(princess_ann, prince_andrew)) ? creep
   Call: (15) insert(princess_ann, [prince_edward], _3980) ? creep
 ^ Call: (16) not(precedes(princess_ann, prince_edward)) ? creep
   Call: (17) precedes(princess_ann, prince_edward) ? creep
   Call: (18) offspring(princess_ann, _27504) ? creep
   Exit: (18) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Call: (18) male(princess_ann) ? creep
   Fail: (18) male(princess_ann) ? creep
   Redo: (17) precedes(princess_ann, prince_edward) ? creep
   Call: (18) offspring(princess_ann, _1764) ? creep
   Exit: (18) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Call: (18) female(princess_ann) ? creep
   Exit: (18) female(princess_ann) ? creep
   Call: (18) female(prince_edward) ? creep
   Fail: (18) female(prince_edward) ? creep
   Redo: (17) precedes(princess_ann, prince_edward) ? creep
   Call: (18) offspring(princess_ann, _9040) ? creep
   Exit: (18) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
   Call: (18) male(princess_ann) ? creep
   Fail: (18) male(princess_ann) ? creep
   Fail: (17) precedes(princess_ann, prince_edward) ? creep
 ^ Exit: (16) not(user:precedes(princess_ann, prince_edward)) ? creep
   Call: (16) insert(princess_ann, [], _136) ? creep
   Exit: (16) insert(princess_ann, [], [princess_ann]) ? creep
   Exit: (15) insert(princess_ann, [prince_edward], [prince_edward, princess_ann]) ? creep
   Exit: (14) insert(princess_ann, [prince_andrew, prince_edward], [prince_andrew, prince_edward, princess_ann]) ? creep
   Exit: (13) sort_succession_list([princess_ann, prince_andrew, prince_edward], [prince_andrew, prince_edward, princess_ann]) ? creep
   Call: (13) insert(prince_charles, [prince_andrew, prince_edward, princess_ann], _18) ? creep
 ^ Call: (14) not(precedes(prince_charles, prince_andrew)) ? creep
   Call: (15) precedes(prince_charles, prince_andrew) ? creep
   Call: (16) offspring(prince_charles, _22054) ? creep
   Exit: (16) offspring(prince_charles, queen_elizabeth) ? creep
   Call: (16) offspring(prince_andrew, queen_elizabeth) ? creep
   Exit: (16) offspring(prince_andrew, queen_elizabeth) ? creep
   Call: (16) male(prince_charles) ? creep
   Exit: (16) male(prince_charles) ? creep
   Call: (16) male(prince_andrew) ? creep
   Exit: (16) male(prince_andrew) ? creep
   Call: (16) older(prince_charles, prince_andrew) ? creep
   Call: (17) older_than(prince_charles, prince_andrew) ? creep
   Fail: (17) older_than(prince_charles, prince_andrew) ? creep
   Redo: (16) older(prince_charles, prince_andrew) ? creep
   Call: (17) older_than(prince_charles, _484) ? creep
   Exit: (17) older_than(prince_charles, princess_ann) ? creep
   Call: (17) older(princess_ann, prince_andrew) ? creep
   Call: (18) older_than(princess_ann, prince_andrew) ? creep
   Exit: (18) older_than(princess_ann, prince_andrew) ? creep
   Exit: (17) older(princess_ann, prince_andrew) ? creep
   Exit: (16) older(prince_charles, prince_andrew) ? creep
   Exit: (15) precedes(prince_charles, prince_andrew) ? creep
 ^ Fail: (14) not(user:precedes(prince_charles, prince_andrew)) ? creep
   Redo: (13) insert(prince_charles, [prince_andrew, prince_edward, princess_ann], _18) ? creep
   Exit: (13) insert(prince_charles, [prince_andrew, prince_edward, princess_ann], [prince_charles, prince_andrew, prince_edward, princess_ann]) ? creep
   Exit: (12) sort_succession_list([prince_charles, princess_ann, prince_andrew, prince_edward], [prince_charles, prince_andrew, prince_edward, princess_ann]) ? creep
   Exit: (11) sortedSuccessionList(queen_elizabeth, [prince_charles, prince_andrew, prince_edward, princess_ann]) ? creep
SuccessionList = [prince_charles, prince_andrew, prince_edward, princess_ann].
```

The line of succession based on the old Royal Succession Rule:
[prince_charles, prince_andrew, prince_edward, princess_ann]

2. Recently, the Royal succession rule has been modified. The throne is now passed down according to the order of birth irrespective of gender. Modify your rules and Prolog knowledge base to handle the new succession rule. Explain the necessary changes to the knowledge needed to represent the new information. Use this new succession rule to determine the new line of succession based on the same knowledge given. Show your results using a trace.

Modifications to the rules from above:
Choose the older child regardless of whether the child is male or female.

New modified rules:
older(A,B):- older_than(A,B).
older(A,B):- older_than(A,X), older(X,B).
precedes(X, Y):- offspring(X, A), offspring(Y, A), older(X, Y), not(queen(X)), not(queen(Y)).

Trace:

```
Call: (11) sortedSuccessionList(queen_elizabeth, _12628) ? creep
Call: (12) findall(_14038, offspring(_14038, queen_elizabeth), _14046) ? creep
Call: (17) offspring(_14038, queen_elizabeth) ? creep
Exit: (17) offspring(prince_charles, queen_elizabeth) ? creep
Redo: (17) offspring(_14038, queen_elizabeth) ? creep
Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
Redo: (17) offspring(_14038, queen_elizabeth) ? creep
Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
Redo: (17) offspring(_14038, queen_elizabeth) ? creep
Exit: (17) offspring(prince_edward, queen_elizabeth) ? creep
Exit: (12) findall(_14038, user:offspring(_14038, queen_elizabeth), [prince_charles, princess_ann, prince_andrew, prince_edward]) ? creep
Call: (12) sort_succession_list([prince_charles, princess_ann, prince_andrew, prince_edward], _12628) ? creep
Call: (13) sort_succession_list([princess_ann, prince_andrew, prince_edward], _23048) ? creep
Call: (14) sort_succession_list([prince_andrew, prince_edward], _23860) ? creep
Call: (15) sort_succession_list([prince_edward], _24672) ? creep
Call: (16) sort_succession_list([], _25484) ? creep
Exit: (16) sort_succession_list([], []) ? creep
Call: (16) insert(prince_edward, [], _24672) ? creep
Exit: (16) insert(prince_edward, [], [prince_edward]) ? creep
Exit: (15) sort_succession_list([prince_edward], [prince_edward]) ? creep
Call: (15) insert(prince_andrew, [prince_edward], _23860) ? creep
Call: (16) not(precedes(prince_andrew, prince_edward)) ? creep
Call: (17) precedes(prince_andrew, prince_edward) ? creep
Call: (18) offspring(prince_andrew, _738) ? creep
Exit: (18) offspring(prince_andrew, queen_elizabeth) ? creep
Call: (18) offspring(prince_edward, queen_elizabeth) ? creep
Exit: (18) offspring(prince_edward, queen_elizabeth) ? creep
Call: (18) older(prince_andrew, prince_edward) ? creep
Call: (19) older_than(prince_andrew, prince_edward) ? creep
Exit: (19) older_than(prince_andrew, prince_edward) ? creep
Exit: (18) older(prince_andrew, prince_edward) ? creep
Call: (18) not(queen(prince_andrew)) ? creep
Call: (19) queen(prince_andrew) ? creep
Fail: (19) queen(prince_andrew) ? creep
Exit: (18) not(user:queen(prince_andrew)) ? creep
Call: (18) not(queen(prince_edward)) ? creep
Call: (19) queen(prince_edward) ? creep
Fail: (19) queen(prince_edward) ? creep
Exit: (18) not(user:queen(prince_edward)) ? creep
Exit: (17) precedes(prince_andrew, prince_edward) ? creep
Fail: (16) not(user:precedes(prince_andrew, prince_edward)) ? creep
Redo: (15) insert(prince_andrew, [prince_edward], _112) ? creep
Exit: (15) insert(prince_andrew, [prince_edward], [prince_andrew, prince_edward]) ? creep
Exit: (14) sort_succession_list([prince_andrew, prince_edward], [prince_andrew, prince_edward]) ? creep
Call: (14) insert(princess_ann, [prince_andrew, prince_edward], _110) ? creep
Call: (15) not(precedes(princess_ann, prince_andrew)) ? creep
Call: (16) precedes(princess_ann, prince_andrew) ? creep
Call: (17) offspring(princess_ann, _20268) ? creep
Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
Call: (17) offspring(prince_andrew, queen_elizabeth) ? creep
Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
Call: (17) older(princess_ann, prince_andrew) ? creep
Call: (18) older_than(princess_ann, prince_andrew) ? creep
Exit: (18) older_than(princess_ann, prince_andrew) ? creep
Exit: (17) older(princess_ann, prince_andrew) ? creep
Call: (17) not(queen(princess_ann)) ? creep
Call: (18) queen(princess_ann) ? creep
Fail: (18) queen(princess_ann) ? creep
Exit: (17) not(user:queen(princess_ann)) ? creep
Call: (17) not(queen(prince_andrew)) ? creep
Call: (18) queen(prince_andrew) ? creep
Fail: (18) queen(prince_andrew) ? creep
Exit: (17) not(user:queen(prince_andrew)) ? creep
Exit: (16) precedes(princess_ann, prince_andrew) ? creep
Fail: (15) not(user:precedes(princess_ann, prince_andrew)) ? creep
Redo: (14) insert(princess_ann, [prince_andrew, prince_edward], _110) ? creep
Exit: (14) insert(princess_ann, [prince_andrew, prince_edward], [princess_ann, prince_andrew, prince_edward]) ? creep
Exit: (13) sort_succession_list([princess_ann, prince_andrew, prince_edward], [princess_ann, prince_andrew, prince_edward]) ? creep
Call: (13) insert(prince_charles, [princess_ann, prince_andrew, prince_edward], _18) ? creep
Call: (14) not(precedes(prince_charles, princess_ann)) ? creep
Call: (15) precedes(prince_charles, princess_ann) ? creep
Call: (16) offspring(prince_charles, _8308) ? creep
Exit: (16) offspring(prince_charles, queen_elizabeth) ? creep

   Call: (16) offspring(prince_charles, _8308) ? creep
   Exit: (16) offspring(prince_charles, queen_elizabeth) ? creep
   Call: (16) offspring(princess_ann, queen_elizabeth) ? skip
   Exit: (16) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (16) older(prince_charles, princess_ann) ? creep
   Call: (17) older_than(prince_charles, princess_ann) ? creep
   Exit: (17) older_than(prince_charles, princess_ann) ? creep
   Exit: (16) older(prince_charles, princess_ann) ? creep
^  Call: (16) not(queen(prince_charles)) ? creep
   Call: (17) queen(prince_charles) ? creep
   Fail: (17) queen(prince_charles) ? creep
^  Exit: (16) not(user:queen(prince_charles)) ? creep
^  Call: (16) not(queen(princess_ann)) ? creep
   Call: (17) queen(princess_ann) ? creep
   Fail: (17) queen(princess_ann) ? creep
^  Exit: (16) not(user:queen(princess_ann)) ? creep
   Exit: (15) precedes(prince_charles, princess_ann) ? creep
^  Fail: (14) not(user:precedes(prince_charles, princess_ann)) ? creep
   Redo: (13) insert(prince_charles, [princess_ann, prince_andrew, prince_edward], _18) ? creep
   Exit: (13) insert(prince_charles, [princess_ann, prince_andrew, prince_edward], [prince_charles, princess_ann, prince_andrew, prince_edward]) ? creep
   Exit: (12) sort_succession_list([prince_charles, princess_ann, prince_andrew, prince_edward], [prince_charles, princess_ann, prince_andrew, prince_edward]) ? creep
   Exit: (11) sortedSuccessionList(queen_elizabeth, [prince_charles, princess_ann, prince_andrew, prince_edward]) ? creep
SuccessionList = [prince_charles, princess_ann, prince_andrew, prince_edward].
```

The line of succession based on the modified Royal Succession Rule:
[prince_charles, princess_ann, prince_andrew, prince_edward]