NANYANG TECHNOLOGICAL UNIVERSITY

CZ3005: Artificial Intelligence

TS6

Assignment 2 Report

Anandarajan Sindini
Kolady Anamika Martin
Verma Nandini

Table of Contents

Exercise 1: The Smart Phone Rivalry

- **First Order Logic (FOL)**
- **Prolog clauses**
- **Query Trace**

Exercise 2.1: The Royal Family

- **First Order Logic (FOL)**
- **Prolog Clauses**
- **Trace for original line of succession**

Exercise 2.2: The Royal Family

- **First Order Logic (FOL)**
- **Prolog Clauses:**
- **Trace for modified line of succession**

Exercise 1: The Smart Phone Rivalry

First Order Logic (FOL)

Assumptions:

1. If X is a competitor of Y, Y is also a competitor of X
2. Stevey is the boss of appy
3. X is a rival of Y does not mean Y is a rival of X

Predicates:

Company(X): X is a company

Competitors(X, Y): X and Y are each other's competitors

Developed(X, A): X developed A

SmartPhoneTechnology(A): A is a smartphone technology

Steal(X, A): X has stolen A

Boss(X, A): X is a boss of company A

Business(B): B is a business

Rival(X, Y): X is a rival of Y

Unethical(X): X is unethical

Statements	First Order Logic (FOL)
sumsum, a competitor of appy	Company(sumsum) Company(appy) Competitors(sumsum, appy)
Sumsum … developed some nice smart phone technology called galactica-s3	Developed(sumsum, galactica-s3) SmartPhoneTechnology(galactica-s3)
galactica-s3 … stolen by stevey	Steal(stevey, galactica-s3)
stevey, who is a boss	Boss(stevey, appy)
unethical for a boss to steal business from rival companies	∀ A. ∀ B. ∀ X. ∀ Y. (Company (A) ∧ Boss(X,A) ∧ Steal(X,B) ∧ Business(B) ∧ Developed(Y,B) ∧ Company(Y) ∧ Rival(X,Y)) → Unethical(X)
A competitor of appy is a rival	∀ X. (Competitor(X, appy) → Rival(X, appy))
Smartphone technology is business	∀ A. (SmartPhoneTechnology(A) → Business(A))

Prolog clauses

```
/* relations */

company(sumsum).
company(appy).
competitors(sumsum, appy).
developed(sumsum, galactica-s3).
smartphonetechnology(galactica-s3).
steal(stevey, galactica-s3).
boss(stevey, appy).

/* rules */

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

business(X):-
        smartphonetechnology(X).

unethical(X):-
        boss(X, A), company(A), steal(X, B), business(B), developed(Y, B), company(Y), rival(Y, A).
```

Query Trace

```
?- [smartPhoneRivalry].
true.

?- unethical(stevey).
true.

?- trace.
true.

[trace]  ?- unethical(stevey).
   Call: (10) unethical(stevey) ? creep
   Call: (11) boss(stevey, _25172) ? creep
   Exit: (11) boss(stevey, appy) ? creep
   Call: (11) company(appy) ? creep
   Exit: (11) company(appy) ? creep
   Call: (11) steal(stevey, _28182) ? creep
   Exit: (11) steal(stevey, galactica-s3) ? creep
   Call: (11) business(galactica-s3) ? creep
   Call: (12) smartphonetechnology(galactica-s3) ? creep
   Exit: (12) smartphonetechnology(galactica-s3) ? creep
   Exit: (11) business(galactica-s3) ? creep
   Call: (11) developed(_32698, galactica-s3) ? creep
   Exit: (11) developed(sumsum, galactica-s3) ? creep
   Call: (11) company(sumsum) ? creep
   Exit: (11) company(sumsum) ? creep
   Call: (11) rival(sumsum, appy) ? creep
   Call: (12) competitors(sumsum, appy) ? creep
   Exit: (12) competitors(sumsum, appy) ? creep
   Exit: (11) rival(sumsum, appy) ? creep
   Exit: (10) unethical(stevey) ? creep
true.
```

Conclusion - Stevey is unethical.

Exercise 2.1: The Royal Family

Assumptions:

1. The order of birth is: Prince Charles, Princess Ann, Prince Andrew, Prince Edward.
2. Throne is passed down to the male children according to order of birth before the female children, in the order of birth.

Predicates:

queen(X): X is the queen

male(X): X is a male

female(X): X is a female

offspring(X,Y): X is an offspring of Y

elder(X,Y): X is elder than Y

successor(X,Y): either X or Y will be the successor to the throne

Explanation:
Since gender of the child affects their position in the line of succession to the throne, two separate rules are needed to determine the order of birth for male children (Rule #1) and female children (Rule #2) respectively.

Based on the above rules, another rule to give precedence to males over females is needed. Hence, Rule #3 is created to choose the male line of succession first before moving to the female line of succession.

First Order Logic (FOL)

Statements	First Order Logic (FOL)
queen elizabeth, the monarch of United Kingdom	queen(queen_elizabeth) female(queen_elizabeth)
four offsprings; namely:- prince charles, princess ann, prince andrew and prince edward	male(prince_charles) female(princess_ann) male(prince_andrew) male(prince_edward) offspring(prince_charles, queen_elizabeth) offspring(princess_ann, queen_elizabeth) offspring(prince_andrew, queen_elizabeth) offspring(prince_edward, queen_elizabeth)
according to the order of birth	elder(prince_charles, princess_ann) elder(prince_charles, prince_andrew) elder(prince_charles, prince_edward) elder(princess_ann, prince_andrew) elder(princess_ann, prince_edward) elder(prince_andrew, prince_edward)
passed down along the male line according to the order of birth before the consideration along the female line	∀ X. ∀ Y. (offspring(X,queen) ∧ offspring(Y,queen) ∧ male(X) ∧ male(Y) ∧ elder(X,Y) → successor(X,Y) ∀ X. ∀ Y. (offspring(X,queen) ∧ offspring(Y,queen) ∧ female(X) ∧ female(Y) ∧ elder(X,Y) ∧ (/∈(queen(X)) ∧ (/∈(queen(Y)) → successor(X,Y) ∀ X. ∀ Y. (offspring(X,queen) ∧ offspring(Y,queen) ∧ male(X) ∧ female(Y) ∧ (/∈(queen(Y)) → successor(X,Y)

Prolog Clauses

```
/* relations */
queen(queen_elizabeth).
female(queen_elizabeth).
female(princess_ann).

male(prince_charles).
male(prince_andrew).
male(prince_edward).

elder(prince_charles, princess_ann).
elder(prince_charles, prince_andrew).
elder(prince_charles, prince_edward).
elder(princess_ann, prince_andrew).
elder(princess_ann, prince_edward).
elder(prince_andrew, prince_edward).

offspring(prince_charles, queen_elizabeth).
offspring(princess_ann, queen_elizabeth).
offspring(prince_andrew, queen_elizabeth).
offspring(prince_edward, queen_elizabeth).
```

```
/* Rules */
/*Rule 1: Older male child precedes younger male child*/
successor(X,Y):-
    offspring(X,A), offspring(Y,A), /*both offsprings having the same parent*/
    male(X), male(Y), elder(X,Y).

/*Rule 2: Older female child precedes younger female child */
successor(X,Y):-
    offspring(X,A), offspring(Y,A), /*both offsprings having the same parent*/
    female(X), female(Y), elder(X,Y),
    not(queen(X)), not(queen(Y)). /*both female compared must not be the queen as queen precedes all offspring*/

/*Rule 3: Male child precedes female child */
successor(X,Y):-
    offspring(X,A), offspring(Y,A), /*both offsprings having same parent */
    male(X), female(Y),
    not(queen(Y)). /*queen precedes offspring*/


/*Sorting*/
insert(X, [Y|Z], [Y|W]):-
    not(successor(X,Y)), !, insert(X, Z, W).

insert(X, Z, [X|Z]).

lineOfSuccesion([X|Y], OrderedLine):-
    lineOfSuccesion(Y, OrderedEnd),insert(X, OrderedEnd, OrderedLine).

lineOfSuccesion([], []).

finalSuccessionLine(X, SuccessionLine):-
    findall(Y, offspring(Y,X), Offspring), lineOfSuccesion(Offspring, SuccessionLine).
```

Trace for original line of succession

```
?- trace,finalSuccessionLine(queen_elizabeth,X).
   Call: (11) finalSuccessionLine(queen_elizabeth, _6036) ? creep
^  Call: (12) findall(_7376, offspring(_7376, queen_elizabeth), _7384) ? creep
   Call: (17) offspring(_7376, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_charles, queen_elizabeth) ? creep
   Redo: (17) offspring(_7376, queen_elizabeth) ? creep
   Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
   Redo: (17) offspring(_7376, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Redo: (17) offspring(_7376, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_edward, queen_elizabeth) ? creep
^  Exit: (12) findall(_7376, user:offspring(_7376, queen_elizabeth), [prince_charles, prin
cess_ann, prince_andrew, prince_edward]) ? creep
   Call: (12) lineOfSuccesion([prince_charles, princess_ann, prince_andrew, prince_edward]
, _6036) ? creep
   Call: (13) lineOfSuccesion([princess_ann, prince_andrew, prince_edward], _15770) ? cree
p
   Call: (14) lineOfSuccesion([prince_andrew, prince_edward], _16526) ? creep
   Call: (15) lineOfSuccesion([prince_edward], _17282) ? creep
   Call: (16) lineOfSuccesion([], _18038) ? creep
   Exit: (16) lineOfSuccesion([], []) ? creep
   Call: (16) insert(prince_edward, [], _17282) ? creep
   Exit: (16) insert(prince_edward, [], [prince_edward]) ? creep
   Exit: (15) lineOfSuccesion([prince_edward], [prince_edward]) ? creep
   Call: (15) insert(prince_andrew, [prince_edward], _16526) ? creep
^  Call: (16) not(successor(prince_andrew, prince_edward)) ? creep
   Call: (17) successor(prince_andrew, prince_edward) ? creep
   Call: (18) offspring(prince_andrew, _24116) ? 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) is_elder(prince_andrew, prince_edward) ? creep
   Call: (19) elder(prince_andrew, prince_edward) ? creep
   Exit: (19) elder(prince_andrew, prince_edward) ? creep
   Exit: (18) is_elder(prince_andrew, prince_edward) ? creep
   Exit: (17) successor(prince_andrew, prince_edward) ? creep
^  Fail: (16) not(user:successor(prince_andrew, prince_edward)) ? creep
   Redo: (15) insert(prince_andrew, [prince_edward], _16526) ? creep
   Exit: (15) insert(prince_andrew, [prince_edward], [prince_andrew, prince_edward]) ? cre
ep
   Exit: (14) lineOfSuccesion([prince_andrew, prince_edward], [prince_andrew, prince_edwar
d]) ? creep
   Call: (14) insert(princess_ann, [prince_andrew, prince_edward], _15770) ? creep
^  Call: (15) not(successor(princess_ann, prince_andrew)) ? creep
   Call: (16) successor(princess_ann, prince_andrew) ? creep

   Call: (17) offspring(princess_ann, _39234) ? 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) successor(princess_ann, prince_andrew) ? creep
   Call: (17) offspring(princess_ann, _44506) ? 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) successor(princess_ann, prince_andrew) ? creep
   Call: (17) offspring(princess_ann, _51278) ? 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) successor(princess_ann, prince_andrew) ? creep
^  Exit: (15) not(user:successor(princess_ann, prince_andrew)) ? creep
   Call: (15) insert(princess_ann, [prince_edward], _37704) ? creep
^  Call: (16) not(successor(princess_ann, prince_edward)) ? creep
   Call: (17) successor(princess_ann, prince_edward) ? creep
   Call: (18) offspring(princess_ann, _59604) ? 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) successor(princess_ann, prince_edward) ? creep
   Call: (18) offspring(princess_ann, _734) ? 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) successor(princess_ann, prince_edward) ? creep
   Call: (18) offspring(princess_ann, _7506) ? 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) successor(princess_ann, prince_edward) ? creep
^  Exit: (16) not(user:successor(princess_ann, prince_edward)) ? creep
   Call: (16) insert(princess_ann, [], _94) ? 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) lineOfSuccesion([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(successor(prince_charles, prince_andrew)) ? creep
   Call: (15) successor(prince_charles, prince_andrew) ? creep
   Call: (16) offspring(prince_charles, _19624) ? 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) is_elder(prince_charles, prince_andrew) ? creep
   Call: (17) elder(prince_charles, prince_andrew) ? creep
   Exit: (17) elder(prince_charles, prince_andrew) ? creep
   Exit: (16) is_elder(prince_charles, prince_andrew) ? creep
   Exit: (15) successor(prince_charles, prince_andrew) ? creep
^  Fail: (14) not(user:successor(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) lineOfSuccesion([prince_charles, princess_ann, prince_andrew, prince_edward]
, [prince_charles, prince_andrew, prince_edward, princess_ann]) ? creep
   Exit: (11) finalSuccessionLine(queen_elizabeth, [prince_charles, prince_andrew, prince_
edward, princess_ann]) ? creep
X = [prince_charles, prince_andrew, prince_edward, princess_ann].
```

Conclusion - [prince_charles, prince_andrew, prince_edward, princess_ann]

Exercise 2.2: The Royal Family

Assumptions:

1. The order of birth is: Prince Charles, Princess Ann, Prince Andrew, Prince Edward.
2. Throne is passed down to the children in the order of birth.

Predicates:

queen(X): X is the queen

male(X): X is a male

female(X): X is a female

offspring(X,Y): X is an offspring of Y

elder(X,Y): X is elder than Y

successor(X,Y): either X or Y will be the successor to the throne

Explanation:
In the modified line of succession, since the throne is given in order of birth irrespective of gender, the rules to determine order of birth for the two genders separately (Rules #1 and #2 in Exercise 2.1) is no longer necessary. Furthermore, Rule #3 which gives males precedence over females is no longer needed.

Therefore, the previous three rules have now been reduced to one rule which simply checks order of birth and then outputs the successor among the two people being compared (Rule #1 in Exercise 2.2).

First Order Logic (FOL)

Statements	First Order Logic (FOL)
queen elizabeth, the monarch of United Kingdom	queen(queen_elizabeth) female(queen_elizabeth)
four offsprings; namely:- prince charles, princess ann, prince andrew and prince edward	male(prince_charles) female(princess_ann) male(prince_andrew) male(prince_edward) offspring(prince_charles, queen_elizabeth) offspring(princess_ann, queen_elizabeth) offspring(prince_andrew, queen_elizabeth) offspring(prince_edward, queen_elizabeth)
according to the order of birth	elder(prince_charles, princess_ann) elder(prince_charles, prince_andrew) elder(prince_charles, prince_edward) elder(princess_ann, prince_andrew) elder(princess_ann, prince_edward) elder(prince_andrew, prince_edward)
the throne is now passed down according to the order of birth irrespective of gender	$\forall$ X. $\forall$ Y. (offspring(X,queen) $\wedge$ offspring(Y,queen) $\wedge$ female(X) $\wedge$ female(Y) $\wedge$ elder(X,Y) $\wedge$ (/$\in$(queen(X)) $\wedge$ (/$\in$(queen(Y)) $\rightarrow$ successor(X,Y)

Prolog Clauses:

```
/* relations */
queen(queen_elizabeth).
female(queen_elizabeth).
female(princess_ann).

male(prince_charles).
male(prince_andrew).
male(prince_edward).

elder(prince_charles, princess_ann).
elder(prince_charles, prince_andrew).
elder(prince_charles, prince_edward).
elder(princess_ann, prince_andrew).
elder(princess_ann, prince_edward).
elder(prince_andrew, prince_edward).

offspring(prince_charles, queen_elizabeth).
offspring(princess_ann, queen_elizabeth).
offspring(prince_andrew, queen_elizabeth).
offspring(prince_edward, queen_elizabeth).
```

```
/* Rules */
/*Rule 1: Older child precedes younger child */
successor(X,Y):-
    offspring(X,A), offspring(Y,A), /*both offsprings having the same parent*/
    elder(X,Y),
    not(queen(X)), not(queen(Y)). /*must not be queen as queen precedes all offspring*/


/*Sorting*/
insert(X, [Y|Z], [Y|W]):-
    not(successor(X,Y)), !, insert(X, Z, W).

insert(X, Z, [X|Z]).

lineOfSuccesion([X|Y], OrderedLine):-
    lineOfSuccesion(Y, OrderedEnd),insert(X, OrderedEnd, OrderedLine).

lineOfSuccesion([], []).

finalSuccessionLine(X, SuccessionLine):-
    findall(Y, offspring(Y,X), Offspring), lineOfSuccesion(Offspring, SuccessionLine).
```

Trace for modified line of succession

```
?- trace,finalSuccessionLine(queen_elizabeth,X).
   Call: (11) finalSuccessionLine(queen_elizabeth, _6000) ? creep
^  Call: (12) findall(_7340, offspring(_7340, queen_elizabeth), _7348) ? creep
   Call: (17) offspring(_7340, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_charles, queen_elizabeth) ? creep
   Redo: (17) offspring(_7340, queen_elizabeth) ? creep
   Exit: (17) offspring(princess_ann, queen_elizabeth) ? creep
   Redo: (17) offspring(_7340, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_andrew, queen_elizabeth) ? creep
   Redo: (17) offspring(_7340, queen_elizabeth) ? creep
   Exit: (17) offspring(prince_edward, queen_elizabeth) ? creep
^  Exit: (12) findall(_7340, user:offspring(_7340, queen_elizabeth), [prince_charles, princes
ss_ann, prince_andrew, prince_edward]) ? creep
   Call: (12) lineOfSuccesion([prince_charles, princess_ann, prince_andrew, prince_edward],
_6000) ? creep
   Call: (13) lineOfSuccesion([princess_ann, prince_andrew, prince_edward], _15734) ? creep
   Call: (14) lineOfSuccesion([prince_andrew, prince_edward], _16490) ? creep
   Call: (15) lineOfSuccesion([prince_edward], _17246) ? creep
   Call: (16) lineOfSuccesion([], _18002) ? creep
   Exit: (16) lineOfSuccesion([], []) ? creep
   Call: (16) insert(prince_edward, [], _17246) ? creep
   Exit: (16) insert(prince_edward, [], [prince_edward]) ? creep
   Exit: (15) lineOfSuccesion([prince_edward], [prince_edward]) ? creep
   Call: (15) insert(prince_andrew, [prince_edward], _16490) ? creep
^  Call: (16) not(successor(prince_andrew, prince_edward)) ? creep
   Call: (17) successor(prince_andrew, prince_edward) ? creep
   Call: (18) offspring(prince_andrew, _24080) ? 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) elder(prince_andrew, prince_edward) ? creep
   Exit: (18) elder(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) successor(prince_andrew, prince_edward) ? creep
^  Fail: (16) not(user:successor(prince_andrew, prince_edward)) ? creep
   Redo: (15) insert(prince_andrew, [prince_edward], _16490) ? creep
   Exit: (15) insert(prince_andrew, [prince_edward], [prince_andrew, prince_edward]) ? creep
   Exit: (14) lineOfSuccesion([prince_andrew, prince_edward], [prince_andrew, prince_edward]
) ? creep

   Call: (14) insert(princess_ann, [prince_andrew, prince_edward], _15734) ? creep
^  Call: (15) not(successor(princess_ann, prince_andrew)) ? creep
   Call: (16) successor(princess_ann, prince_andrew) ? creep
   Call: (17) offspring(princess_ann, _40758) ? 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) elder(princess_ann, prince_andrew) ? creep
   Exit: (17) elder(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(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) successor(princess_ann, prince_andrew) ? creep
^  Fail: (15) not(user:successor(princess_ann, prince_andrew)) ? creep
   Redo: (14) insert(princess_ann, [prince_andrew, prince_edward], _15734) ? creep
   Exit: (14) insert(princess_ann, [prince_andrew, prince_edward], [princess_ann, prince_and
rew, prince_edward]) ? creep
   Exit: (13) lineOfSuccesion([princess_ann, prince_andrew, prince_edward], [princess_ann, p
rince_andrew, prince_edward]) ? creep
   Call: (13) insert(prince_charles, [princess_ann, prince_andrew, prince_edward], _6000) ?
creep
^  Call: (14) not(successor(prince_charles, princess_ann)) ? creep
   Call: (15) successor(prince_charles, princess_ann) ? creep
   Call: (16) offspring(prince_charles, _57436) ? creep
   Exit: (16) offspring(prince_charles, queen_elizabeth) ? creep
   Call: (16) offspring(princess_ann, queen_elizabeth) ? creep
   Exit: (16) offspring(princess_ann, queen_elizabeth) ? creep
   Call: (16) elder(prince_charles, princess_ann) ? creep
   Exit: (16) elder(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) successor(prince_charles, princess_ann) ? creep
^  Fail: (14) not(user:successor(prince_charles, princess_ann)) ? creep
   Redo: (13) insert(prince_charles, [princess_ann, prince_andrew, prince_edward], _18) ? cr
eep
   Exit: (13) insert(prince_charles, [princess_ann, prince_andrew, prince_edward], [prince_c
harles, princess_ann, prince_andrew, prince_edward]) ? creep
   Exit: (12) lineOfSuccesion([prince_charles, princess_ann, prince_andrew, prince_edward],
[prince_charles, princess_ann, prince_andrew, prince_edward]) ? creep
   Exit: (11) finalSuccessionLine(queen_elizabeth, [prince_charles, princess_ann, prince_and
rew, prince_edward]) ? creep
X = [prince_charles, princess_ann, prince_andrew, prince_edward].
```

Conclusion - [prince_charles, princess_ann, prince_andrew, prince_edward]