SC3000 Assignment 2

Lab Group A26

Ananthan Srinath Adhvait -- U2121974B

Tan Wei Yuan -- U2122007H

Ong Tsien Jin -- U2120567H

Table of contents

Exercise 1: The Smart Phone Rivalry

> `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 of `appy`. It is unethical for a boss to steal business from rival companies. A competitor of is a rival. Smart phone technology is business.

1. Translation of natural language to First Order Logic (FOL)

> `sumsum`, a competitor of `appy`

> A competitor of is a rival.

This statement implies that both `sumsum` and `appy` are companies within the same industry. In the real world, not all companies are competitors to each other; however, this question only states two companies, thus we will be making the assumption that all companies are competitors with each other.

$$\forall_x \forall_y \ (Company(x) \land Company(y) \rightarrow Competitor(x, y))$$

Furthermore, since competitors are rivals with each other, we can deduce the following equivilance relation.

$$\forall_x \forall_y \ (Competitor(x, y) \iff Rival(x, y))$$

> ...developed some nice smart phone technology called `galactica-s3`...

A company can develop smart phone technology, but not all the smart phone technology used by the company is developed from the same company.

$$\exists_x \exists_t \ (Company(x) \ \land \ SmartPhoneTechnology(t) \rightarrow Develop(x, t))$$

> Smart phone technology is business.

We can further deduce a logical statement that all smart phone technology is business.

$$\forall_t \ (SmartPhoneTechnology(t) \rightarrow Business(t))$$

> ...all of which was stolen by `stevey`...

We can infer that `stevey` is a person. This suggests that a person can steal technology.

$$\exists_t \exists_h \ (SmartPhoneTechnology(t) \ \land \ Person(h) \rightarrow Steals(h, t))$$

> ... `stevey`, who is a boss of `appy`.

From the earlier deduction that `stevey` is a person, this informs us that the boss of `appy` (a company) is a person. Thus, for every company, there is a person as its boss.

$$\forall_x \left(Company(x) \rightarrow (\exists_h \, Person(h) \land Boss(x, h))\right)$$

> It is unethical for a boss to steal businsess from rival companies.

A boss is a person, who is the boss of a company. This statement suggests that it is only unethical if the two companies involved are rivals.

We can break this statement down into smaller logical statements to follow. Statements (2) and (3) are transitive, as the boss is a person, and a person can steal technology.

1. There are two companies that are rivals.
2. The boss of one of these companies is a person.
3. The boss can steal busines from rival companies.
4. Stealing business from rival companies is always unethical.

Therefore these statements can be expressed in natural language -- there are two companies which are rivals, where the boss of one company steals business from the rival company; this is unethical.

$$\exists_x \exists_y \exists_t \exists_h \, (Boss(x, h) \, \land \, Rival(x, y) \, \land \, Steals(h, t) \, \land \, Develop(y, t) \rightarrow \neg ethical)$$

2. Translating FOL Statements to Prolog

Declaration of Facts

We first begin with declarative statements for the given facts.

```
% Defining companies
company(sumsum).
company(appy).

% Beep boop technology
smartphonetechnology(galactica-s3).

% Develop tech relations
develop(sumsum, galactica-s3) :- company(sumsum), smartphonetechnology(gala

% Stevey is a bad guy
person(stevey).
boss(appy, stevey) :- company(appy), person(stevey).
steals(stevey, galactica-s3) :- person(stevey), business(galactica-s3).
```

Translations

Defining competitors and rivals.

$$\forall_x \forall_y \ (Company(x) \land Company(y) \rightarrow Competitor(x, y))$$
$$\forall_x \forall_y \ (Competitor(x, y) \iff Rival(x, y))$$

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

Defining the definition that smart phone technology is business.

$$\forall_t \ (SmartPhoneTechnology(t) \rightarrow Business(t))$$

```
business(X) :- smartphonetechnology(X).
```

Defining the predicate `not_ethical`.

$$\exists_x \exists_y \exists_t \exists_h$$
$$(Boss(x, h) \ \land \ Rival(x, y) \ \land \ Steals(h, t) \ \land \ Develop(y, t) \rightarrow \neg ethical)$$

```
not_ethical(H) :-
  rival(X,Y),
  boss(X,H),
  develop(Y,T),
  steals(H,T).
```

Prolog Knowledge Base

Complete knowledge base containing the statements from above.

```
% Defining companies
company(sumsum).
company(appy).

% Beep boop technology
smartphonetechnology(galactica-s3).

business(X) :- smartphonetechnology(X).

% Defines relations between companies
competitor(X,Y) :- company(X), company(Y).
rival(X,Y) :- competitor(X,Y).

% Develop tech relations
```

```
develop(sumsum, galactica-s3) :-
    company(sumsum),
    smartphonetechnology(galactica-s3).

% Stevey is a bad guy
person(stevey).
boss(appy, stevey) :- company(appy), person(stevey).
steals(stevey, galactica-s3) :- person(stevey), business(galactica-s3).

% Determining predicate
not_ethical(H) :-
  rival(X,Y),
  boss(X,H),
  develop(Y,T),
  steals(H,T).
```

3. Trace Results

```
?- not_ethical(X).
X = stevey ;
false.

?- trace.
true.

[trace]  ?- not_ethical(X).
   Call: (10) not_ethical(_11536) ? creep
   Call: (11) rival(_12820, _12822) ? creep
   Call: (12) competitor(_12820, _12822) ? creep
   Call: (13) company(_12820) ? creep
   Exit: (13) company(sumsum) ? creep
   Call: (13) company(_12822) ? creep
   Exit: (13) company(sumsum) ? creep
   Exit: (12) competitor(sumsum, sumsum) ? creep
   Exit: (11) rival(sumsum, sumsum) ? creep
   Call: (11) boss(sumsum, _11536) ? creep
   Fail: (11) boss(sumsum, _11536) ? creep
   Redo: (13) company(_12822) ? creep
   Exit: (13) company(appy) ? creep
   Exit: (12) competitor(sumsum, appy) ? creep
   Exit: (11) rival(sumsum, appy) ? creep
   Call: (11) boss(sumsum, _11536) ? creep
   Fail: (11) boss(sumsum, _11536) ? creep
   Redo: (13) company(_12820) ? creep
   Exit: (13) company(appy) ? creep
   Call: (13) company(_12822) ? creep
   Exit: (13) company(sumsum) ? creep
   Exit: (12) competitor(appy, sumsum) ? creep
   Exit: (11) rival(appy, sumsum) ? creep
```

```
   Call: (11) boss(appy, _11536) ? creep
   Call: (12) company(appy) ? creep
   Exit: (12) company(appy) ? creep
   Call: (12) person(stevey) ? creep
   Exit: (12) person(stevey) ? creep
   Exit: (11) boss(appy, stevey) ? creep
   Call: (11) develop(sumsum, _80) ? creep
   Call: (12) company(sumsum) ? creep
   Exit: (12) company(sumsum) ? creep
   Call: (12) smartphonetechnology(galactica-s3) ? creep
   Exit: (12) smartphonetechnology(galactica-s3) ? creep
   Exit: (11) develop(sumsum, galactica-s3) ? creep
   Call: (11) steals(stevey, galactica-s3) ? creep
   Call: (12) person(stevey) ? creep
   Exit: (12) person(stevey) ? creep
   Call: (12) business(galactica-s3) ? creep
   Call: (13) smartphonetechnology(galactica-s3) ? creep
   Exit: (13) smartphonetechnology(galactica-s3) ? creep
   Exit: (12) business(galactica-s3) ? creep
   Exit: (11) steals(stevey, galactica-s3) ? creep
   Exit: (10) not_ethical(stevey) ? creep
X = stevey .
```

Exercise 2: The Royal Family

> 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.

Part 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.

Prolog Rules in Knowledge Base

We approached this problem by validating the n^{th} item and the $(n+1)^{th}$ item in the array.

1. If they are the same gender, then the n^{th} person must be born before the $(n+1)^{th}$ person.
2. If they are different genders, then the n^{th} person must be a male, and the $(n+1)^{th}$ person must be a female.
3. Both the n^{th} and the $(n+1)^{th}$ person must not be crowned.

```
% Define gender
female(queen_elizabeth).
female(princess_ann).
male(prince_charles).
male(prince_andrew).
male(prince_edward).

% Define current crrown
crown(queen_elizabeth).

% Define parent relationships
parent(queen_elizabeth, prince_charles).
parent(queen_elizabeth, princess_ann).
parent(queen_elizabeth, prince_andrew).
parent(queen_elizabeth, prince_edward).

% Define birth order
% born_after(X,Y) --> X is born before Y --> X is older than Y
born_after(queen_elizabeth, prince_charles).
```

```
born_after(prince_charles, princess_ann).
born_after(princess_ann, prince_andrew).
born_after(prince_andrew, prince_edward).

% Determining birth born_after transitivity
born_after_x(A,B) :- born_after(A,B).

born_after_x(A,C) :-
  born_after(A,B), born_after_x(B,C).

% Checks if same gender
same_gender(X,Y) :- (male(X), male(Y)); (female(X), female(Y)).

% Ensures same gender, and second arg is younger sibling
same_g_younger(X,Y) :- same_gender(X,Y), born_after_x(X, Y).

% Ensures that the first arg is male and second arg is female.
diff_g(X,Y) :- male(X), female(Y).

% Main method to verify length and then calls order(X).
get_order(List, Number) :- length(List, Number), order(List).

% Checks for the valid order of the n^th item and the (n+1)^th item
order([H,S|T]) :- order_r(H, [S,T]).
order([H,T]) :- order_r(H, [T]).
order([_]).
order([]).

% Ensures that n^th and (n+1)^th follows the royal rule
order_r_helper(X,Y) :-
    (same_g_younger(X,Y); diff_g(X,Y)),
    not(crown(X)),
    not(crown(Y)).

% Recursive call for (n+1)^th item onwards
order_r(H, [S,T]) :-
  order_r_helper(H,S), order_r(S,T).

order_r(H,T) :- order_r_helper(H, T).
```

Trace Results

```
?- get_order(X, 4).
X = [prince_charles, prince_andrew, prince_edward, princess_ann] .

?- trace.
true.

[trace]  ?- get_order(X, 4).
   Call: (10) get_order(_12954, 4) ? creep
```

```
Call: (11) length(_12954, 4) ? creep
Exit: (11) length([_15066, _15072, _15078, _15084], 4) ? creep
Call: (11) order([_15066, _15072, _15078, _15084]) ? creep
Call: (12) order_r(_15066, [_15072, [_15078, _15084]]) ? creep
Call: (13) order_r_helper(_15066, _15072) ? creep
Call: (14) same_g_younger(_15066, _15072) ? creep
Call: (15) same_gender(_15066, _15072) ? creep
Call: (16) male(_15066) ? creep
Exit: (16) male(prince_charles) ? creep
Call: (16) male(_15072) ? creep
Exit: (16) male(prince_charles) ? creep
Exit: (15) same_gender(prince_charles, prince_charles) ? creep
Call: (15) born_after_x(prince_charles, prince_charles) ? creep
Call: (16) born_after(prince_charles, prince_charles) ? creep
Fail: (16) born_after(prince_charles, prince_charles) ? creep
Redo: (15) born_after_x(prince_charles, prince_charles) ? creep
Call: (16) born_after(prince_charles, _27230) ? creep
Exit: (16) born_after(prince_charles, princess_ann) ? creep
Call: (16) born_after_x(princess_ann, prince_charles) ? creep
Call: (17) born_after(princess_ann, prince_charles) ? creep
Fail: (17) born_after(princess_ann, prince_charles) ? creep
Redo: (16) born_after_x(princess_ann, prince_charles) ? creep
Call: (17) born_after(princess_ann, _32092) ? creep
Exit: (17) born_after(princess_ann, prince_andrew) ? creep
Call: (17) born_after_x(prince_andrew, prince_charles) ? creep
Call: (18) born_after(prince_andrew, prince_charles) ? creep
Fail: (18) born_after(prince_andrew, prince_charles) ? creep
Redo: (17) born_after_x(prince_andrew, prince_charles) ? creep
Call: (18) born_after(prince_andrew, _1566) ? creep
Exit: (18) born_after(prince_andrew, prince_edward) ? creep
Call: (18) born_after_x(prince_edward, prince_charles) ? creep
Call: (19) born_after(prince_edward, prince_charles) ? creep
Fail: (19) born_after(prince_edward, prince_charles) ? creep
Redo: (18) born_after_x(prince_edward, prince_charles) ? creep
Call: (19) born_after(prince_edward, _6428) ? creep
Fail: (19) born_after(prince_edward, _6428) ? creep
Fail: (18) born_after_x(prince_edward, prince_charles) ? creep
Fail: (17) born_after_x(prince_andrew, prince_charles) ? creep
Fail: (16) born_after_x(princess_ann, prince_charles) ? creep
Fail: (15) born_after_x(prince_charles, prince_charles) ? creep
Redo: (16) male(_84) ? creep
Exit: (16) male(prince_andrew) ? creep
Exit: (15) same_gender(prince_charles, prince_andrew) ? creep
Call: (15) born_after_x(prince_charles, prince_andrew) ? creep
Call: (16) born_after(prince_charles, prince_andrew) ? creep
Fail: (16) born_after(prince_charles, prince_andrew) ? creep
Redo: (15) born_after_x(prince_charles, prince_andrew) ? creep
Call: (16) born_after(prince_charles, _16952) ? creep
Exit: (16) born_after(prince_charles, princess_ann) ? creep
Call: (16) born_after_x(princess_ann, prince_andrew) ? creep
Call: (17) born_after(princess_ann, prince_andrew) ? creep
```

```
   Exit: (17) born_after(princess_ann, prince_andrew) ? creep
   Exit: (16) born_after_x(princess_ann, prince_andrew) ? creep
   Exit: (15) born_after_x(prince_charles, prince_andrew) ? creep
   Exit: (14) same_g_younger(prince_charles, prince_andrew) ? creep
^  Call: (14) not(crown(prince_charles)) ? creep
   Call: (15) crown(prince_charles) ? creep
   Fail: (15) crown(prince_charles) ? creep
^  Exit: (14) not(user:crown(prince_charles)) ? creep
^  Call: (14) not(crown(prince_andrew)) ? creep
   Call: (15) crown(prince_andrew) ? creep
   Fail: (15) crown(prince_andrew) ? creep
^  Exit: (14) not(user:crown(prince_andrew)) ? creep
   Exit: (13) order_r_helper(prince_charles, prince_andrew) ? creep
   Call: (13) order_r(prince_andrew, [_90, _96]) ? creep
   Call: (14) order_r_helper(prince_andrew, _90) ? creep
   Call: (15) same_g_younger(prince_andrew, _90) ? creep
   Call: (16) same_gender(prince_andrew, _90) ? creep
   Call: (17) male(prince_andrew) ? creep
   Exit: (17) male(prince_andrew) ? creep
   Call: (17) male(_90) ? creep
   Exit: (17) male(prince_charles) ? creep
   Exit: (16) same_gender(prince_andrew, prince_charles) ? creep
   Call: (16) born_after_x(prince_andrew, prince_charles) ? creep
   Call: (17) born_after(prince_andrew, prince_charles) ? creep
   Fail: (17) born_after(prince_andrew, prince_charles) ? creep
   Redo: (16) born_after_x(prince_andrew, prince_charles) ? creep
   Call: (17) born_after(prince_andrew, _5772) ? creep
   Exit: (17) born_after(prince_andrew, prince_edward) ? creep
   Call: (17) born_after_x(prince_edward, prince_charles) ? creep
   Call: (18) born_after(prince_edward, prince_charles) ? creep
   Fail: (18) born_after(prince_edward, prince_charles) ? creep
   Redo: (17) born_after_x(prince_edward, prince_charles) ? creep
   Call: (18) born_after(prince_edward, _10634) ? creep
   Fail: (18) born_after(prince_edward, _10634) ? creep
   Fail: (17) born_after_x(prince_edward, prince_charles) ? creep
   Fail: (16) born_after_x(prince_andrew, prince_charles) ? creep
   Redo: (17) male(_90) ? creep
   Exit: (17) male(prince_andrew) ? creep
   Exit: (16) same_gender(prince_andrew, prince_andrew) ? creep
   Call: (16) born_after_x(prince_andrew, prince_andrew) ? creep
   Call: (17) born_after(prince_andrew, prince_andrew) ? creep
   Fail: (17) born_after(prince_andrew, prince_andrew) ? creep
   Redo: (16) born_after_x(prince_andrew, prince_andrew) ? creep
   Call: (17) born_after(prince_andrew, _19538) ? creep
   Exit: (17) born_after(prince_andrew, prince_edward) ? creep
   Call: (17) born_after_x(prince_edward, prince_andrew) ? creep
   Call: (18) born_after(prince_edward, prince_andrew) ? creep
   Fail: (18) born_after(prince_edward, prince_andrew) ? creep
   Redo: (17) born_after_x(prince_edward, prince_andrew) ? creep
   Call: (18) born_after(prince_edward, _24400) ? creep
   Fail: (18) born_after(prince_edward, _24400) ? creep
```

```
   Fail: (17) born_after_x(prince_edward, prince_andrew) ? creep
   Fail: (16) born_after_x(prince_andrew, prince_andrew) ? creep
   Redo: (17) male(_90) ? creep
   Exit: (17) male(prince_edward) ? creep
   Exit: (16) same_gender(prince_andrew, prince_edward) ? creep
   Call: (16) born_after_x(prince_andrew, prince_edward) ? creep
   Call: (17) born_after(prince_andrew, prince_edward) ? creep
   Exit: (17) born_after(prince_andrew, prince_edward) ? creep
   Exit: (16) born_after_x(prince_andrew, prince_edward) ? creep
   Exit: (15) same_g_younger(prince_andrew, prince_edward) ? creep
^  Call: (15) not(crown(prince_andrew)) ? creep
   Call: (16) crown(prince_andrew) ? creep
   Fail: (16) crown(prince_andrew) ? creep
^  Exit: (15) not(user:crown(prince_andrew)) ? creep
^  Call: (15) not(crown(prince_edward)) ? creep
   Call: (16) crown(prince_edward) ? creep
   Fail: (16) crown(prince_edward) ? creep
^  Exit: (15) not(user:crown(prince_edward)) ? creep
   Exit: (14) order_r_helper(prince_andrew, prince_edward) ? creep
   Call: (14) order_r(prince_edward, _96) ? creep
   Call: (15) order_r_helper(prince_edward, _6624) ? creep
   Call: (16) same_g_younger(prince_edward, _6624) ? creep
   Call: (17) same_gender(prince_edward, _6624) ? creep
   Call: (18) male(prince_edward) ? creep
   Exit: (18) male(prince_edward) ? creep
   Call: (18) male(_6624) ? creep
   Exit: (18) male(prince_charles) ? creep
   Exit: (17) same_gender(prince_edward, prince_charles) ? creep
   Call: (17) born_after_x(prince_edward, prince_charles) ? creep
   Call: (18) born_after(prince_edward, prince_charles) ? creep
   Fail: (18) born_after(prince_edward, prince_charles) ? creep
   Redo: (17) born_after_x(prince_edward, prince_charles) ? creep
   Call: (18) born_after(prince_edward, _16338) ? creep
   Fail: (18) born_after(prince_edward, _16338) ? creep
   Fail: (17) born_after_x(prince_edward, prince_charles) ? creep
   Redo: (18) male(_6624) ? creep
   Exit: (18) male(prince_andrew) ? creep
   Exit: (17) same_gender(prince_edward, prince_andrew) ? creep
   Call: (17) born_after_x(prince_edward, prince_andrew) ? creep
   Call: (18) born_after(prince_edward, prince_andrew) ? creep
   Fail: (18) born_after(prince_edward, prince_andrew) ? creep
   Redo: (17) born_after_x(prince_edward, prince_andrew) ? creep
   Call: (18) born_after(prince_edward, _24432) ? creep
   Fail: (18) born_after(prince_edward, _24432) ? creep
   Fail: (17) born_after_x(prince_edward, prince_andrew) ? creep
   Redo: (18) male(_6624) ? creep
   Exit: (18) male(prince_edward) ? creep
   Exit: (17) same_gender(prince_edward, prince_edward) ? creep
   Call: (17) born_after_x(prince_edward, prince_edward) ? creep
   Call: (18) born_after(prince_edward, prince_edward) ? creep
   Fail: (18) born_after(prince_edward, prince_edward) ? creep
```

```
   Redo: (17) born_after_x(prince_edward, prince_edward) ? creep
   Call: (18) born_after(prince_edward, _32526) ? creep
   Fail: (18) born_after(prince_edward, _32526) ? creep
   Fail: (17) born_after_x(prince_edward, prince_edward) ? creep
   Redo: (17) same_gender(prince_edward, _6624) ? creep
   Call: (18) female(prince_edward) ? creep
   Fail: (18) female(prince_edward) ? creep
   Fail: (17) same_gender(prince_edward, _116) ? creep
   Fail: (16) same_g_younger(prince_edward, _116) ? creep
   Redo: (15) order_r_helper(prince_edward, _116) ? creep
   Call: (16) diff_g(prince_edward, _116) ? creep
   Call: (17) male(prince_edward) ? creep
   Exit: (17) male(prince_edward) ? creep
   Call: (17) female(_116) ? creep
   Exit: (17) female(queen_elizabeth) ? creep
   Exit: (16) diff_g(prince_edward, queen_elizabeth) ? creep
^  Call: (16) not(crown(prince_edward)) ? creep
   Call: (17) crown(prince_edward) ? creep
   Fail: (17) crown(prince_edward) ? creep
^  Exit: (16) not(user:crown(prince_edward)) ? creep
^  Call: (16) not(crown(queen_elizabeth)) ? creep
   Call: (17) crown(queen_elizabeth) ? creep
   Exit: (17) crown(queen_elizabeth) ? creep
^  Fail: (16) not(user:crown(queen_elizabeth)) ? creep
   Redo: (17) female(_116) ? creep
   Exit: (17) female(princess_ann) ? creep
   Exit: (16) diff_g(prince_edward, princess_ann) ? creep
^  Call: (16) not(crown(prince_edward)) ? creep
   Call: (17) crown(prince_edward) ? creep
   Fail: (17) crown(prince_edward) ? creep
^  Exit: (16) not(user:crown(prince_edward)) ? creep
^  Call: (16) not(crown(princess_ann)) ? creep
   Call: (17) crown(princess_ann) ? creep
   Fail: (17) crown(princess_ann) ? creep
^  Exit: (16) not(user:crown(princess_ann)) ? creep
   Exit: (15) order_r_helper(prince_edward, princess_ann) ? creep
   Call: (15) order_r(princess_ann, _122) ? creep
   Call: (16) order_r_helper(princess_ann, _26090) ? creep
   Call: (17) same_g_younger(princess_ann, _26090) ? creep
   Call: (18) same_gender(princess_ann, _26090) ? creep
   Call: (19) male(princess_ann) ? creep
   Fail: (19) male(princess_ann) ? creep
   Redo: (18) same_gender(princess_ann, _26090) ? creep
   Call: (19) female(princess_ann) ? creep
   Exit: (19) female(princess_ann) ? creep
   Call: (19) female(_26090) ? creep
   Exit: (19) female(queen_elizabeth) ? creep
   Exit: (18) same_gender(princess_ann, queen_elizabeth) ? creep
   Call: (18) born_after_x(princess_ann, queen_elizabeth) ? creep
   Call: (19) born_after(princess_ann, queen_elizabeth) ? creep
   Fail: (19) born_after(princess_ann, queen_elizabeth) ? creep
```

```
Redo: (18) born_after_x(princess_ann, queen_elizabeth) ? creep
Call: (19) born_after(princess_ann, _2570) ? creep
Exit: (19) born_after(princess_ann, prince_andrew) ? creep
Call: (19) born_after_x(prince_andrew, queen_elizabeth) ? creep
Call: (20) born_after(prince_andrew, queen_elizabeth) ? creep
Fail: (20) born_after(prince_andrew, queen_elizabeth) ? creep
Redo: (19) born_after_x(prince_andrew, queen_elizabeth) ? creep
Call: (20) born_after(prince_andrew, _7432) ? creep
Exit: (20) born_after(prince_andrew, prince_edward) ? creep
Call: (20) born_after_x(prince_edward, queen_elizabeth) ? creep
Call: (21) born_after(prince_edward, queen_elizabeth) ? creep
Fail: (21) born_after(prince_edward, queen_elizabeth) ? creep
Redo: (20) born_after_x(prince_edward, queen_elizabeth) ? creep
Call: (21) born_after(prince_edward, _12294) ? creep
Fail: (21) born_after(prince_edward, _12294) ? creep
Fail: (20) born_after_x(prince_edward, queen_elizabeth) ? creep
Fail: (19) born_after_x(prince_andrew, queen_elizabeth) ? creep
Fail: (18) born_after_x(princess_ann, queen_elizabeth) ? creep
Redo: (19) female(_128) ? creep
Exit: (19) female(princess_ann) ? creep
Exit: (18) same_gender(princess_ann, princess_ann) ? creep
Call: (18) born_after_x(princess_ann, princess_ann) ? creep
Call: (19) born_after(princess_ann, princess_ann) ? creep
Fail: (19) born_after(princess_ann, princess_ann) ? creep
Redo: (18) born_after_x(princess_ann, princess_ann) ? creep
Call: (19) born_after(princess_ann, _22008) ? creep
Exit: (19) born_after(princess_ann, prince_andrew) ? creep
Call: (19) born_after_x(prince_andrew, princess_ann) ? creep
Call: (20) born_after(prince_andrew, princess_ann) ? creep
Fail: (20) born_after(prince_andrew, princess_ann) ? creep
Redo: (19) born_after_x(prince_andrew, princess_ann) ? creep
Call: (20) born_after(prince_andrew, _26870) ? creep
Exit: (20) born_after(prince_andrew, prince_edward) ? creep
Call: (20) born_after_x(prince_edward, princess_ann) ? creep
Call: (21) born_after(prince_edward, princess_ann) ? creep
Fail: (21) born_after(prince_edward, princess_ann) ? creep
Redo: (20) born_after_x(prince_edward, princess_ann) ? creep
Call: (21) born_after(prince_edward, _31732) ? creep
Fail: (21) born_after(prince_edward, _31732) ? creep
Fail: (20) born_after_x(prince_edward, princess_ann) ? creep
Fail: (19) born_after_x(prince_andrew, princess_ann) ? creep
Fail: (18) born_after_x(princess_ann, princess_ann) ? creep
Fail: (17) same_g_younger(princess_ann, _128) ? creep
Redo: (16) order_r_helper(princess_ann, _128) ? creep
Call: (17) diff_g(princess_ann, _128) ? creep
Call: (18) male(princess_ann) ? creep
Fail: (18) male(princess_ann) ? creep
Fail: (17) diff_g(princess_ann, _128) ? creep
Fail: (16) order_r_helper(princess_ann, _128) ? creep
Redo: (15) order_r(princess_ann, _122) ? creep
Call: (16) order_r_helper(princess_ann, _122) ? creep
```

```
Call: (17) same_g_younger(princess_ann, _122) ? creep
Call: (18) same_gender(princess_ann, _122) ? creep
Call: (19) male(princess_ann) ? creep
Fail: (19) male(princess_ann) ? creep
Redo: (18) same_gender(princess_ann, _122) ? creep
Call: (19) female(princess_ann) ? creep
Exit: (19) female(princess_ann) ? creep
Call: (19) female(_122) ? creep
Exit: (19) female(queen_elizabeth) ? creep
Exit: (18) same_gender(princess_ann, queen_elizabeth) ? creep
Call: (18) born_after_x(princess_ann, queen_elizabeth) ? creep
Call: (19) born_after(princess_ann, queen_elizabeth) ? creep
Fail: (19) born_after(princess_ann, queen_elizabeth) ? creep
Redo: (18) born_after_x(princess_ann, queen_elizabeth) ? creep
Call: (19) born_after(princess_ann, _18738) ? creep
Exit: (19) born_after(princess_ann, prince_andrew) ? creep
Call: (19) born_after_x(prince_andrew, queen_elizabeth) ? creep
Call: (20) born_after(prince_andrew, queen_elizabeth) ? creep
Fail: (20) born_after(prince_andrew, queen_elizabeth) ? creep
Redo: (19) born_after_x(prince_andrew, queen_elizabeth) ? creep
Call: (20) born_after(prince_andrew, _23600) ? creep
Exit: (20) born_after(prince_andrew, prince_edward) ? creep
Call: (20) born_after_x(prince_edward, queen_elizabeth) ? creep
Call: (21) born_after(prince_edward, queen_elizabeth) ? creep
Fail: (21) born_after(prince_edward, queen_elizabeth) ? creep
Redo: (20) born_after_x(prince_edward, queen_elizabeth) ? creep
Call: (21) born_after(prince_edward, _28462) ? creep
Fail: (21) born_after(prince_edward, _28462) ? creep
Fail: (20) born_after_x(prince_edward, queen_elizabeth) ? creep
Fail: (19) born_after_x(prince_andrew, queen_elizabeth) ? creep
Fail: (18) born_after_x(princess_ann, queen_elizabeth) ? creep
Redo: (19) female(_122) ? creep
Exit: (19) female(princess_ann) ? creep
Exit: (18) same_gender(princess_ann, princess_ann) ? creep
Call: (18) born_after_x(princess_ann, princess_ann) ? creep
Call: (19) born_after(princess_ann, princess_ann) ? creep
Fail: (19) born_after(princess_ann, princess_ann) ? creep
Redo: (18) born_after_x(princess_ann, princess_ann) ? creep
Call: (19) born_after(princess_ann, _2552) ? creep
Exit: (19) born_after(princess_ann, prince_andrew) ? creep
Call: (19) born_after_x(prince_andrew, princess_ann) ? creep
Call: (20) born_after(prince_andrew, princess_ann) ? creep
Fail: (20) born_after(prince_andrew, princess_ann) ? creep
Redo: (19) born_after_x(prince_andrew, princess_ann) ? creep
Call: (20) born_after(prince_andrew, _7414) ? creep
Exit: (20) born_after(prince_andrew, prince_edward) ? creep
Call: (20) born_after_x(prince_edward, princess_ann) ? creep
Call: (21) born_after(prince_edward, princess_ann) ? creep
Fail: (21) born_after(prince_edward, princess_ann) ? creep
Redo: (20) born_after_x(prince_edward, princess_ann) ? creep
Call: (21) born_after(prince_edward, _12276) ? creep
```

```
Fail: (21) born_after(prince_edward, _12276) ? creep
Fail: (20) born_after_x(prince_edward, princess_ann) ? creep
Fail: (19) born_after_x(prince_andrew, princess_ann) ? creep
Fail: (18) born_after_x(princess_ann, princess_ann) ? creep
Fail: (17) same_g_younger(princess_ann, _122) ? creep
Redo: (16) order_r_helper(princess_ann, _122) ? creep
Call: (17) diff_g(princess_ann, _122) ? creep
Call: (18) male(princess_ann) ? creep
Fail: (18) male(princess_ann) ? creep
Fail: (17) diff_g(princess_ann, _122) ? creep
Fail: (16) order_r_helper(princess_ann, _122) ? creep
Fail: (15) order_r(princess_ann, _122) ? creep
Redo: (14) order_r(prince_edward, _96) ? creep
Call: (15) order_r_helper(prince_edward, _96) ? creep
Call: (16) same_g_younger(prince_edward, _96) ? creep
Call: (17) same_gender(prince_edward, _96) ? creep
Call: (18) male(prince_edward) ? creep
Exit: (18) male(prince_edward) ? creep
Call: (18) male(_96) ? creep
Exit: (18) male(prince_charles) ? creep
Exit: (17) same_gender(prince_edward, prince_charles) ? creep
Call: (17) born_after_x(prince_edward, prince_charles) ? creep
Call: (18) born_after(prince_edward, prince_charles) ? creep
Fail: (18) born_after(prince_edward, prince_charles) ? creep
Redo: (17) born_after_x(prince_edward, prince_charles) ? creep
Call: (18) born_after(prince_edward, _33314) ? creep
Fail: (18) born_after(prince_edward, _33314) ? creep
Fail: (17) born_after_x(prince_edward, prince_charles) ? creep
Redo: (18) male(_96) ? creep
Exit: (18) male(prince_andrew) ? creep
Exit: (17) same_gender(prince_edward, prince_andrew) ? creep
Call: (17) born_after_x(prince_edward, prince_andrew) ? creep
Call: (18) born_after(prince_edward, prince_andrew) ? creep
Fail: (18) born_after(prince_edward, prince_andrew) ? creep
Redo: (17) born_after_x(prince_edward, prince_andrew) ? creep
Call: (18) born_after(prince_edward, _5772) ? creep
Fail: (18) born_after(prince_edward, _5772) ? creep
Fail: (17) born_after_x(prince_edward, prince_andrew) ? creep
Redo: (18) male(_96) ? creep
Exit: (18) male(prince_edward) ? creep
Exit: (17) same_gender(prince_edward, prince_edward) ? creep
Call: (17) born_after_x(prince_edward, prince_edward) ? creep
Call: (18) born_after(prince_edward, prince_edward) ? creep
Fail: (18) born_after(prince_edward, prince_edward) ? creep
Redo: (17) born_after_x(prince_edward, prince_edward) ? creep
Call: (18) born_after(prince_edward, _13866) ? creep
Fail: (18) born_after(prince_edward, _13866) ? creep
Fail: (17) born_after_x(prince_edward, prince_edward) ? creep
Redo: (17) same_gender(prince_edward, _96) ? creep
Call: (18) female(prince_edward) ? creep
Fail: (18) female(prince_edward) ? creep
```

```
   Fail: (17) same_gender(prince_edward, _96) ? creep
   Fail: (16) same_g_younger(prince_edward, _96) ? creep
   Redo: (15) order_r_helper(prince_edward, _96) ? creep
   Call: (16) diff_g(prince_edward, _96) ? creep
   Call: (17) male(prince_edward) ? creep
   Exit: (17) male(prince_edward) ? creep
   Call: (17) female(_96) ? creep
   Exit: (17) female(queen_elizabeth) ? creep
   Exit: (16) diff_g(prince_edward, queen_elizabeth) ? creep
^  Call: (16) not(crown(prince_edward)) ? creep
   Call: (17) crown(prince_edward) ? creep
   Fail: (17) crown(prince_edward) ? creep
^  Exit: (16) not(user:crown(prince_edward)) ? creep
^  Call: (16) not(crown(queen_elizabeth)) ? creep
   Call: (17) crown(queen_elizabeth) ? creep
   Exit: (17) crown(queen_elizabeth) ? creep
^  Fail: (16) not(user:crown(queen_elizabeth)) ? creep
   Redo: (17) female(_96) ? creep
   Exit: (17) female(princess_ann) ? creep
   Exit: (16) diff_g(prince_edward, princess_ann) ? creep
^  Call: (16) not(crown(prince_edward)) ? creep
   Call: (17) crown(prince_edward) ? creep
   Fail: (17) crown(prince_edward) ? creep
^  Exit: (16) not(user:crown(prince_edward)) ? creep
^  Call: (16) not(crown(princess_ann)) ? creep
   Call: (17) crown(princess_ann) ? creep
   Fail: (17) crown(princess_ann) ? creep
^  Exit: (16) not(user:crown(princess_ann)) ? creep
   Exit: (15) order_r_helper(prince_edward, princess_ann) ? creep
   Exit: (14) order_r(prince_edward, princess_ann) ? creep
   Exit: (13) order_r(prince_andrew, [prince_edward, princess_ann]) ? creep
   Exit: (12) order_r(prince_charles, [prince_andrew, [prince_edward, prince
   Exit: (11) order([prince_charles, prince_andrew, prince_edward, princess_
   Exit: (10) get_order([prince_charles, prince_andrew, prince_edward, princ
X = [prince_charles, prince_andrew, prince_edward, princess_ann] .
```

Part 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.

Changes Required to Knowledge Base

In order to disregard the rule where male heirs come before female heirs, we can simplify the predicates in the Royal succession rule.

The original Royal succession rule is pivotal on the following predicate where either:

1. The n^{th} person is the same gender and born before the $(n+1)^{th}$ person.
2. If they are of differnt genders, the n^{th} person must be male and the $(n+1)^{th}$ person must be female.

```
% Ensures that n^th and (n+1)^th follows the royal rule
order_r_helper(X,Y) :-
    (same_g_younger(X,Y); diff_g(X,Y)),
    not(crown(X)),
    not(crown(Y)).
```

This can be modified to ensure transitivity from the n^{th} person to the $(n+1)^{th}$ person, where the former is born prior to the latter -- regardless of gender.

This predicate is already defined (and utilised) in the previous part.

```
% Determining birth born_after transitivity
born_after_x(A,B) :- born_after(A,B).

born_after_x(A,C) :-
  born_after(A,B), born_after_x(B,C).
```

As such, the new updated predicate can be written as follows.

```
% Ensures that n^th and (n+1)^th follows the updated rule
order_r_helper(X,Y) :-
    born_after_x(X,Y),
```

```
    not(crown(X)),
    not(crown(Y)).
```

The updated knowledge base is as follows.

```
% Define gender
female(queen_elizabeth).
female(princess_ann).
male(prince_charles).
male(prince_andrew).
male(prince_edward).

% Define current crrown
crown(queen_elizabeth).

% Define parent relationships
parent(queen_elizabeth, prince_charles).
parent(queen_elizabeth, princess_ann).
parent(queen_elizabeth, prince_andrew).
parent(queen_elizabeth, prince_edward).

% Define birth order
% born_after(X,Y) --> X is born before Y --> X is older than Y
born_after(queen_elizabeth, prince_charles).
born_after(prince_charles, princess_ann).
born_after(princess_ann, prince_andrew).
born_after(prince_andrew, prince_edward).

% Determining birth born_after transitivity
born_after_x(A,B) :- born_after(A,B).

born_after_x(A,C) :-
  born_after(A,B), born_after_x(B,C).

% Checks if same gender
same_gender(X,Y) :- (male(X), male(Y)); (female(X), female(Y)).

% Ensures same gender, and second arg is younger sibling
same_g_younger(X,Y) :- same_gender(X,Y), born_after_x(X, Y).

% Ensures that the first arg is male and second arg is female.
diff_g(X,Y) :- male(X), female(Y).

% Main method to verify length and then calls order(X).
get_order(List, Number) :- length(List, Number), order(List).

% Checks for the valid order of the n^th item and the (n+1)^th item
order([H,S|T]) :- order_r(H, [S,T]).
order([H,T]) :- order_r(H, [T]).
order([X]) :- not(crown(X)).
```

```
order([]).

% Ensures that n^th and (n+1)^th follows the royal rule
order_r_helper(X,Y) :-
    born_after_x(X,Y),
    not(crown(X)),
    not(crown(Y)).

% Recursive call for (n+1)^th item onwards
order_r(H, [S,T]) :-
  order_r_helper(H,S), order_r(S,T).

order_r(H,T) :- order_r_helper(H, T).
```

Trace Results

```
?- get_order(X, 4).
X = [prince_charles, princess_ann, prince_andrew, prince_edward] .

?- trace.
true.

[trace]  ?- get_order(X, 4).
   Call: (10) get_order(_28228, 4) ? creep
   Call: (11) length(_28228, 4) ? creep
   Exit: (11) length([_30340, _30346, _30352, _30358], 4) ? creep
   Call: (11) order([_30340, _30346, _30352, _30358]) ? creep
   Call: (12) order_r(_30340, [_30346, [_30352, _30358]]) ? creep
   Call: (13) order_r_helper(_30340, _30346) ? creep
   Call: (14) born_after_x(_30340, _30346) ? creep
   Call: (15) born_after(_30340, _30346) ? creep
   Exit: (15) born_after(queen_elizabeth, prince_charles) ? creep
   Exit: (14) born_after_x(queen_elizabeth, prince_charles) ? creep
^  Call: (14) not(crown(queen_elizabeth)) ? creep
   Call: (15) crown(queen_elizabeth) ? creep
   Exit: (15) crown(queen_elizabeth) ? creep
^  Fail: (14) not(user:crown(queen_elizabeth)) ? creep
   Redo: (15) born_after(_30340, _30346) ? creep
   Exit: (15) born_after(prince_charles, princess_ann) ? creep
   Exit: (14) born_after_x(prince_charles, princess_ann) ? creep
^  Call: (14) not(crown(prince_charles)) ? creep
   Call: (15) crown(prince_charles) ? creep
   Fail: (15) crown(prince_charles) ? creep
^  Exit: (14) not(user:crown(prince_charles)) ? creep
^  Call: (14) not(crown(princess_ann)) ? creep
   Call: (15) crown(princess_ann) ? creep
   Fail: (15) crown(princess_ann) ? creep
^  Exit: (14) not(user:crown(princess_ann)) ? creep
   Exit: (13) order_r_helper(prince_charles, princess_ann) ? creep
   Call: (13) order_r(princess_ann, [_30352, _30358]) ? creep
```

```
   Call: (14) order_r_helper(princess_ann, _30352) ? creep
   Call: (15) born_after_x(princess_ann, _30352) ? creep
   Call: (16) born_after(princess_ann, _30352) ? creep
   Exit: (16) born_after(princess_ann, prince_andrew) ? creep
   Exit: (15) born_after_x(princess_ann, prince_andrew) ? creep
^  Call: (15) not(crown(princess_ann)) ? creep
   Call: (16) crown(princess_ann) ? creep
   Fail: (16) crown(princess_ann) ? creep
^  Exit: (15) not(user:crown(princess_ann)) ? creep
^  Call: (15) not(crown(prince_andrew)) ? creep
   Call: (16) crown(prince_andrew) ? creep
   Fail: (16) crown(prince_andrew) ? creep
^  Exit: (15) not(user:crown(prince_andrew)) ? creep
   Exit: (14) order_r_helper(princess_ann, prince_andrew) ? creep
   Call: (14) order_r(prince_andrew, _30358) ? creep
   Call: (15) order_r_helper(prince_andrew, _62862) ? creep
   Call: (16) born_after_x(prince_andrew, _62862) ? creep
   Call: (17) born_after(prince_andrew, _62862) ? creep
   Exit: (17) born_after(prince_andrew, prince_edward) ? creep
   Exit: (16) born_after_x(prince_andrew, prince_edward) ? creep
^  Call: (16) not(crown(prince_andrew)) ? creep
   Call: (17) crown(prince_andrew) ? creep
   Fail: (17) crown(prince_andrew) ? creep
^  Exit: (16) not(user:crown(prince_andrew)) ? creep
^  Call: (16) not(crown(prince_edward)) ? creep
   Call: (17) crown(prince_edward) ? creep
   Fail: (17) crown(prince_edward) ? creep
^  Exit: (16) not(user:crown(prince_edward)) ? creep
   Exit: (15) order_r_helper(prince_andrew, prince_edward) ? creep
   Call: (15) order_r(prince_edward, _120) ? creep
   Call: (16) order_r_helper(prince_edward, _6640) ? creep
   Call: (17) born_after_x(prince_edward, _6640) ? creep
   Call: (18) born_after(prince_edward, _6640) ? creep
   Fail: (18) born_after(prince_edward, _6640) ? creep
   Redo: (17) born_after_x(prince_edward, _6640) ? creep
   Call: (18) born_after(prince_edward, _10700) ? creep
   Fail: (18) born_after(prince_edward, _10700) ? creep
   Fail: (17) born_after_x(prince_edward, _6640) ? creep
   Fail: (16) order_r_helper(prince_edward, _6640) ? creep
   Redo: (15) order_r(prince_edward, _120) ? creep
   Call: (16) order_r_helper(prince_edward, _120) ? creep
   Call: (17) born_after_x(prince_edward, _120) ? creep
   Call: (18) born_after(prince_edward, _120) ? creep
   Fail: (18) born_after(prince_edward, _120) ? creep
   Redo: (17) born_after_x(prince_edward, _120) ? creep
   Call: (18) born_after(prince_edward, _18802) ? creep
   Fail: (18) born_after(prince_edward, _18802) ? creep
   Fail: (17) born_after_x(prince_edward, _120) ? creep
   Fail: (16) order_r_helper(prince_edward, _120) ? creep
   Fail: (15) order_r(prince_edward, _120) ? creep
   Redo: (16) born_after_x(prince_andrew, _114) ? creep
```

```
   Call: (17) born_after(prince_andrew, _23664) ? creep
   Exit: (17) born_after(prince_andrew, prince_edward) ? creep
   Call: (17) born_after_x(prince_edward, _114) ? creep
   Call: (18) born_after(prince_edward, _114) ? creep
   Fail: (18) born_after(prince_edward, _114) ? creep
   Redo: (17) born_after_x(prince_edward, _114) ? creep
   Call: (18) born_after(prince_edward, _28526) ? creep
   Fail: (18) born_after(prince_edward, _28526) ? creep
   Fail: (17) born_after_x(prince_edward, _114) ? creep
   Fail: (16) born_after_x(prince_andrew, _114) ? creep
   Fail: (15) order_r_helper(prince_andrew, _114) ? creep
   Redo: (14) order_r(prince_andrew, _96) ? creep
   Call: (15) order_r_helper(prince_andrew, _96) ? creep
   Call: (16) born_after_x(prince_andrew, _96) ? creep
   Call: (17) born_after(prince_andrew, _96) ? creep
   Exit: (17) born_after(prince_andrew, prince_edward) ? creep
   Exit: (16) born_after_x(prince_andrew, prince_edward) ? creep
^  Call: (16) not(crown(prince_andrew)) ? creep
   Call: (17) crown(prince_andrew) ? creep
   Fail: (17) crown(prince_andrew) ? creep
^  Exit: (16) not(user:crown(prince_andrew)) ? creep
^  Call: (16) not(crown(prince_edward)) ? creep
   Call: (17) crown(prince_edward) ? creep
   Fail: (17) crown(prince_edward) ? creep
^  Exit: (16) not(user:crown(prince_edward)) ? creep
   Exit: (15) order_r_helper(prince_andrew, prince_edward) ? creep
   Exit: (14) order_r(prince_andrew, prince_edward) ? creep
   Exit: (13) order_r(princess_ann, [prince_andrew, prince_edward]) ? creep
   Exit: (12) order_r(prince_charles, [princess_ann, [prince_andrew, prince_
   Exit: (11) order([prince_charles, princess_ann, prince_andrew, prince_edw
   Exit: (10) get_order([prince_charles, princess_ann, prince_andrew, prince
X = [prince_charles, princess_ann, prince_andrew, prince_edward] .
```