NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

SC3000 - Artificial Intelligence

Assignment 2 - Learning to Use Prolog as a Logic Programming Tool

Lab Group: SCSB

Group: Group 14

Members	**Matric No.**
Lester Kwek Zuo Wei	U2120699C
Kiew Ten Wei	U2221627A
Donna Chua Jie Ning	U2121254E

Exercise 1: The Smart Phone Rivalry

sumsum, a competitor of appy, developed some nice smart phone technology called galacticas3, 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 is a rival. Smart phone technology is business.

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

- Assume if X is a competitor of Y, Y is a competitor of X

Natural Language	First Order Logic(FOL)
Sumsum developed Galacticas3.	developed(sumsum, galacticas3)
Stevey stole Galacticas3 and Stevey is a boss of Appy.	stolen(stevey, galacticas3) ∧ boss(stevey, appy)
If a boss steals business from a rival company, it is unethical.	∀x,∀y,∀z (boss(x,y) ∧ stolen(x,z) ∧ business(z)∧ rival(w,y) → unethical(x))
Competitors are rivals	∀x,∀y (competitor(x,y) → rival(x,y))
Smartphone technology counts as business.	∀x (smartphoneTech(x)→business(x))

2. Write these FOL statements as Prolog clauses.

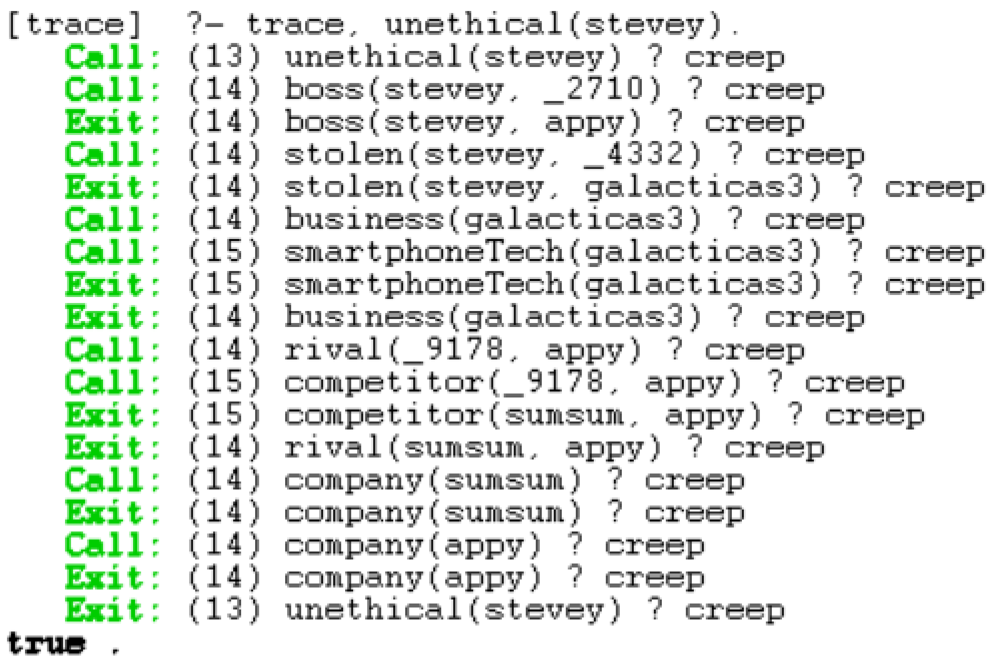
```
% Entity Definitions
company(sumsum).
company(appy).

% Fact Definitions
developed(sumsum, galacticas3).
stolen(stevey, galacticas3).
boss(stevey, appy).
competitor(sumsum, appy).
smartphoneTech(galacticas3).

% Derived facts and rules
business(X) :- smartphoneTech(X).  % Defines that all smartphone
technology is business

% Defines rivals based on competitor relationships, bidirectional
rival(X, Y) :- competitor(X, Y).
rival(Y, X) :- competitor(X, Y).
```

```
% Ethical Evaluation Rule
unethical(X) :-
    boss(X, Y),
    stolen(X, Z),
    business(Z),
    rival(W, Y),
    company(W),
    company(Y).
```

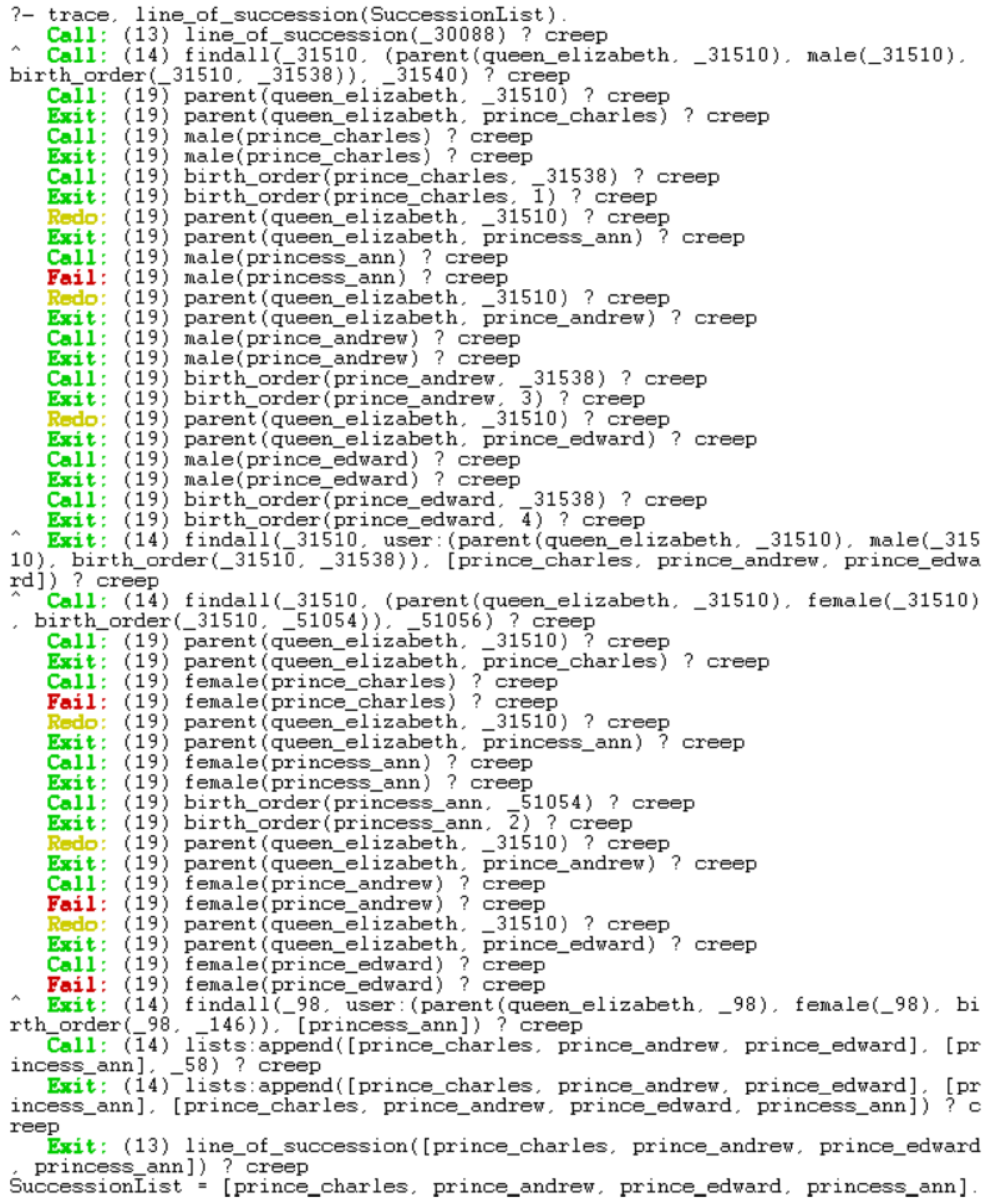


```
[trace]  ?- trace, unethical(stevey).
   Call: (13) unethical(stevey) ? creep
   Call: (14) boss(stevey, _2710) ? creep
   Exit: (14) boss(stevey, appy) ? creep
   Call: (14) stolen(stevey, _4332) ? creep
   Exit: (14) stolen(stevey, galacticas3) ? creep
   Call: (14) business(galacticas3) ? creep
   Call: (15) smartphoneTech(galacticas3) ? creep
   Exit: (15) smartphoneTech(galacticas3) ? creep
   Exit: (14) business(galacticas3) ? creep
   Call: (14) rival(_9178, appy) ? creep
   Call: (15) competitor(_9178, appy) ? creep
   Exit: (15) competitor(sumsum, appy) ? creep
   Exit: (14) rival(sumsum, appy) ? creep
   Call: (14) company(sumsum) ? creep
   Exit: (14) company(sumsum) ? creep
   Call: (14) company(appy) ? creep
   Exit: (14) company(appy) ? creep
   Exit: (13) unethical(stevey) ? creep
true .
```

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.

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.



```
?- trace, line_of_succession(SuccessionList).
   Call: (13) line_of_succession(_30088) ? creep
^  Call: (14) findall(_31510, (parent(queen_elizabeth, _31510), male(_31510),
birth_order(_31510, _31538)), _31540) ? creep
   Call: (19) parent(queen_elizabeth, _31510) ? creep
   Exit: (19) parent(queen_elizabeth, prince_charles) ? creep
   Call: (19) male(prince_charles) ? creep
   Exit: (19) male(prince_charles) ? creep
   Call: (19) birth_order(prince_charles, _31538) ? creep
   Exit: (19) birth_order(prince_charles, 1) ? creep
   Redo: (19) parent(queen_elizabeth, _31510) ? creep
   Exit: (19) parent(queen_elizabeth, princess_ann) ? creep
   Call: (19) male(princess_ann) ? creep
   Fail: (19) male(princess_ann) ? creep
   Redo: (19) parent(queen_elizabeth, _31510) ? creep
   Exit: (19) parent(queen_elizabeth, prince_andrew) ? creep
   Call: (19) male(prince_andrew) ? creep
   Exit: (19) male(prince_andrew) ? creep
   Call: (19) birth_order(prince_andrew, _31538) ? creep
   Exit: (19) birth_order(prince_andrew, 3) ? creep
   Redo: (19) parent(queen_elizabeth, _31510) ? creep
   Exit: (19) parent(queen_elizabeth, prince_edward) ? creep
   Call: (19) male(prince_edward) ? creep
   Exit: (19) male(prince_edward) ? creep
   Call: (19) birth_order(prince_edward, _31538) ? creep
   Exit: (19) birth_order(prince_edward, 4) ? creep
^  Exit: (14) findall(_31510, user:(parent(queen_elizabeth, _31510), male(_315
10), birth_order(_31510, _31538)), [prince_charles, prince_andrew, prince_edwa
rd]) ? creep
^  Call: (14) findall(_31510, (parent(queen_elizabeth, _31510), female(_31510)
, birth_order(_31510, _51054)), _51056) ? creep
   Call: (19) parent(queen_elizabeth, _31510) ? creep
   Exit: (19) parent(queen_elizabeth, prince_charles) ? creep
   Call: (19) female(prince_charles) ? creep
   Fail: (19) female(prince_charles) ? creep
   Redo: (19) parent(queen_elizabeth, _31510) ? creep
   Exit: (19) parent(queen_elizabeth, princess_ann) ? creep
   Call: (19) female(princess_ann) ? creep
   Exit: (19) female(princess_ann) ? creep
   Call: (19) birth_order(princess_ann, _51054) ? creep
   Exit: (19) birth_order(princess_ann, 2) ? creep
   Redo: (19) parent(queen_elizabeth, _31510) ? creep
   Exit: (19) parent(queen_elizabeth, prince_andrew) ? creep
   Call: (19) female(prince_andrew) ? creep
   Fail: (19) female(prince_andrew) ? creep
   Redo: (19) parent(queen_elizabeth, _31510) ? creep
   Exit: (19) parent(queen_elizabeth, prince_edward) ? creep
   Call: (19) female(prince_edward) ? creep
   Fail: (19) female(prince_edward) ? creep
^  Exit: (14) findall(_98, user:(parent(queen_elizabeth, _98), female(_98), bi
rth_order(_98, _146)), [princess_ann]) ? creep
   Call: (14) lists:append([prince_charles, prince_andrew, prince_edward], [pr
incess_ann], _58) ? creep
   Exit: (14) lists:append([prince_charles, prince_andrew, prince_edward], [pr
incess_ann], [prince_charles, prince_andrew, prince_edward, princess_ann]) ? c
reep
   Exit: (13) line_of_succession([prince_charles, prince_andrew, prince_edward
, princess_ann]) ? creep
SuccessionList = [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.

Old Rules:

Collecting Male Successors:

```
findall(Name, (
    parent(queen_elizabeth, Name),
    male(Name),
    birth_order(Name, _)
), MaleSuccessors),
```

Logic: Collects all male children of Queen Elizabeth.

Collecting Female Successors:

```
findall(Name, (
    parent(queen_elizabeth, Name),
    female(Name),
    birth_order(Name, _)
), FemaleSuccessors),
```

Logic: Collects all female children of Queen Elizabeth, similar to males.

Concatenating Lists:

```
append(MaleSuccessors, FemaleSuccessors, SuccessionList).
```

Logic: Combines the list of male successors with the list of female successors.

Modified Rules:

Collecting All Children with Birth Order:

```
findall(Order-Name, (
    parent(queen_elizabeth, Name),
    birth_order(Name, Order)
), NamesAndOrders),
```

Logic: Collects all children of Queen Elizabeth, pairing each child with their birth order.

Sorting by Birth Order:

```
keysort(NamesAndOrders, SortedNamesAndOrders),
```

Logic: Uses the keysort/2 function to sort children based on the key, which in this case is the birth order.

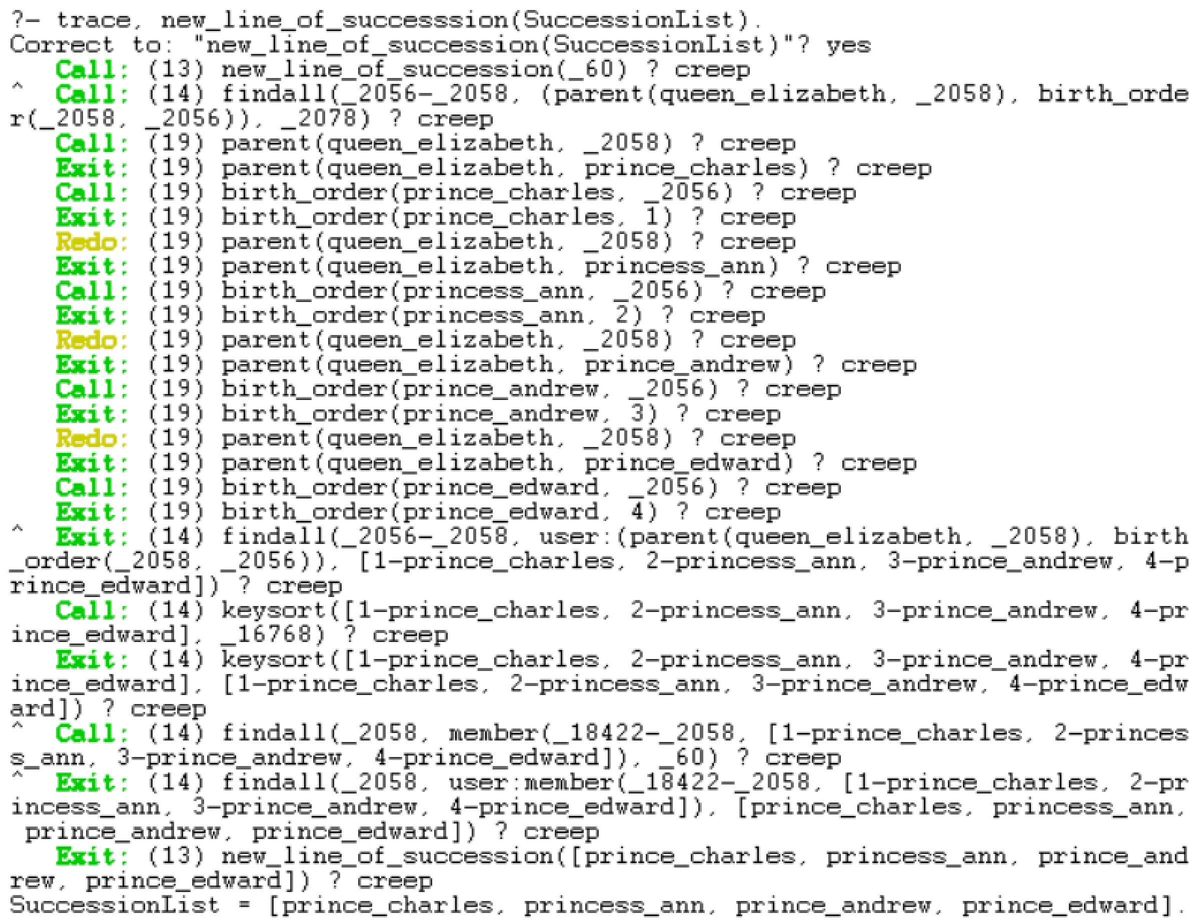
Extracting Names from Sorted Pairs:

```
findall(Name, member(_Order-Name, SortedNamesAndOrders),
SuccessionList).
```

Logic: Extracts just the names from the sorted Order-Name pairs to create the final list of successors.

Explanation:

The revised regulation gathers and arranges applicants only according to birth order, eliminating the gender-based division seen in the previous rule.
From Manual Concatenation to Automated Sorting: The improved rule automates sorting according to an objective criterion (birth order), eliminating the need to manually attach men before females.



```
?- trace, new_line_of_successsion(SuccessionList).
Correct to: "new_line_of_succession(SuccessionList)"? yes
   Call: (13) new_line_of_succession(_60) ? creep
^  Call: (14) findall(_2056-_2058, (parent(queen_elizabeth, _2058), birth_orde
r(_2058, _2056)), _2078) ? creep
   Call: (19) parent(queen_elizabeth, _2058) ? creep
   Exit: (19) parent(queen_elizabeth, prince_charles) ? creep
   Call: (19) birth_order(prince_charles, _2056) ? creep
   Exit: (19) birth_order(prince_charles, 1) ? creep
   Redo: (19) parent(queen_elizabeth, _2058) ? creep
   Exit: (19) parent(queen_elizabeth, princess_ann) ? creep
   Call: (19) birth_order(princess_ann, _2056) ? creep
   Exit: (19) birth_order(princess_ann, 2) ? creep
   Redo: (19) parent(queen_elizabeth, _2058) ? creep
   Exit: (19) parent(queen_elizabeth, prince_andrew) ? creep
   Call: (19) birth_order(prince_andrew, _2056) ? creep
   Exit: (19) birth_order(prince_andrew, 3) ? creep
   Redo: (19) parent(queen_elizabeth, _2058) ? creep
   Exit: (19) parent(queen_elizabeth, prince_edward) ? creep
   Call: (19) birth_order(prince_edward, _2056) ? creep
   Exit: (19) birth_order(prince_edward, 4) ? creep
^  Exit: (14) findall(_2056-_2058, user:(parent(queen_elizabeth, _2058), birth
_order(_2058, _2056)), [1-prince_charles, 2-princess_ann, 3-prince_andrew, 4-p
rince_edward]) ? creep
   Call: (14) keysort([1-prince_charles, 2-princess_ann, 3-prince_andrew, 4-pr
ince_edward], _16768) ? creep
   Exit: (14) keysort([1-prince_charles, 2-princess_ann, 3-prince_andrew, 4-pr
ince_edward], [1-prince_charles, 2-princess_ann, 3-prince_andrew, 4-prince_edw
ard]) ? creep
^  Call: (14) findall(_2058, member(_18422-_2058, [1-prince_charles, 2-princes
s_ann, 3-prince_andrew, 4-prince_edward]), _60) ? creep
^  Exit: (14) findall(_2058, user:member(_18422-_2058, [1-prince_charles, 2-pr
incess_ann, 3-prince_andrew, 4-prince_edward]), [prince_charles, princess_ann,
 prince_andrew, prince_edward]) ? creep
   Exit: (13) new_line_of_succession([prince_charles, princess_ann, prince_and
rew, prince_edward]) ? creep
SuccessionList = [prince_charles, princess_ann, prince_andrew, prince_edward].
```