SC3000: Artificial Intelligence
Lab Assignment 2 Report

Name	Email	Matric Number
Pu Fanyi	FPU001@e.ntu.edu.sg	U2220175K
Ting Ruo Chee	RTING002@e.ntu.edu.sg	U2220572C
Soo Ying Xi	D220001@e.ntu.edu.sg	U2220021D

Tutorial Group: SCS2
Teaching Assistant: Wang Zeyu

School of Computer Science and Engineering
Nanyang Technological University

2023/2024 Semester 2

Exercise 1: The Smart Phone Rivalry

Q1.1

Facts:
CompetitorOf(Sumsum, Appy)
DevelopedBy(Galactica-s3, Sumsum)
IsSmartPhone(Galactica-s3)
Stole(Stevey, Galactica-s3)
BossOf(Stevey, Appy)

Rules:
∀boss, ∀company, ∀tech, ∀rival, [BossOf(boss, company) ∧ Stole(boss, tech) ∧ Business(tech) ∧ DevelopedBy(tech, rival) ∧ RivalOf(rival, company)] ⇒ Unethical(boss)
∀x, ∀y CompetitorOf(x, y) ⇔ RivalOf(x, y)
∀x, ∀y, DevelopedBy(y,x) ∧ IsSmartPhone(y) ⇒ Business(x)

Q1.2

Ruo Chee:

```
competitor(sumsum, appy).
developed(galacticas3, sumsum).
smartphone(galacticas3).
stole(stevey, galacticas3).
boss(stevey, appy).
unethical(Boss) :- boss(Boss, Company), stole(Boss, Tech),
business(Rival), developed(Tech, Rival), rival(Rival, Company).
rival(X, Y) :- competitor(X, Y).
business(X) :- smartphone(X), developed(Y, X).
```

Q1.3

```
?- trace, unethical(stevey).
   Call: (13) unethical(stevey) ? creep
   Call: (14) boss(stevey, _64676) ? creep
   Exit: (14) boss(stevey, appy) ? creep
   Call: (14) stole(stevey, _66298) ? creep
   Exit: (14) stole(stevey, galacticas3) ? creep
   Call: (14) business(_67920) ? creep
   Call: (15) smartphone(_68728) ? creep
   Exit: (15) smartphone(galacticas3) ? creep
   Call: (15) developed(galacticas3, _67920) ? creep
   Exit: (15) developed(galacticas3, sumsum) ? creep
   Exit: (14) business(sumsum) ? creep
   Call: (14) developed(galacticas3, sumsum) ? creep
   Exit: (14) developed(galacticas3, sumsum) ? creep
   Call: (14) rival(sumsum, appy) ? creep
   Call: (15) competitor(sumsum, appy) ? creep
   Exit: (15) competitor(sumsum, appy) ? creep
   Exit: (14) rival(sumsum, appy) ? creep
   Exit: (13) unethical(stevey) ? creep
true.
```

Exercise 2: The Royal Family

Note: We implemented two versions of solution. The first one does not use the sorting function, while the second one uses the sorting function.

Q2.1

First method:

```
% Facts about the offspring of Queen Elizabeth
offspring(prince_charles, male, 1).
offspring(princess_ann, female, 2).
offspring(prince_andrew, male, 3).
offspring(prince_edward, male, 4).

% Helper rule to compare offspring for sorting
compare_offspring(Order, X, Y) :-
    offspring(X, GenderX, OrderX),
    offspring(Y, GenderY, OrderY),
    (   GenderX = GenderY -> compare(Order, OrderX, OrderY)
    ;   GenderX = male -> Order = <
    ;   Order = >
    ).

% Determine the line of succession based on the old rule
old_line_of_succession(SuccessionList) :-
    findall(Name, offspring(Name, _, _), Offspring),
    predsort(compare_offspring, Offspring, SuccessionList).
```

SWI-Prolog (AMD64, Multi-threaded, version 9.2.3)

File Edit Settings Run Debug Help

```
% c:/Users/pufan/Desktop/a/a.pl compiled 0.00 sec, 6 clauses
?- trace, old_line_of_succession(X).
   Call: (13) old_line_of_succession(_20324) ? creep
^  Call: (14) findall(_21736, offspring(_21736, _21742, _21744), _21746) ? creep
   Call: (18) offspring(_21736, _21742, _21744) ? creep
   Exit: (18) offspring(prince_charles, male, 1) ? creep
   Redo: (18) offspring(_21736, _21742, _21744) ? creep
   Exit: (18) offspring(princess_ann, female, 2) ? creep
   Redo: (18) offspring(_21736, _21742, _21744) ? creep
   Exit: (18) offspring(prince_andrew, male, 3) ? creep
   Redo: (18) offspring(_21736, _21742, _21744) ? creep
   Exit: (18) offspring(prince_edward, male, 4) ? creep
^  Exit: (14) findall(_21736, user:offspring(_21736, _21742, _21744), [prince_charles, princess_ann, prince_andrew, prince_edward]) ? creep
^  Call: (14) sort:predsort(compare_offspring, [prince_charles, princess_ann, prince_andrew, prince_edward], _20324) ? creep
   Call: (17) compare_offspring(_32356, prince_charles, princess_ann) ? creep
   Call: (18) offspring(prince_charles, _33172, _33174) ? creep
   Exit: (18) offspring(prince_charles, male, 1) ? creep
   Call: (18) offspring(princess_ann, _34804, _34806) ? creep
   Exit: (18) offspring(princess_ann, female, 2) ? creep
   Call: (18) male=female ? creep
   Fail: (18) male=female ? creep
   Redo: (17) compare_offspring(_32356, prince_charles, princess_ann) ? creep
   Call: (18) male=male ? creep
   Exit: (18) male=male ? creep
   Call: (18) _32356=(<) ? creep
   Exit: (18) (<)=(<) ? creep
   Exit: (17) compare_offspring(<, prince_charles, princess_ann) ? creep
   Call: (17) compare_offspring(_42940, prince_andrew, prince_edward) ? creep
   Call: (18) offspring(prince_andrew, _43756, _43758) ? creep
   Exit: (18) offspring(prince_andrew, male, 3) ? creep
   Call: (18) offspring(prince_edward, _45388, _45390) ? creep
   Exit: (18) offspring(prince_edward, male, 4) ? creep
   Call: (18) male=male ? creep
   Exit: (18) male=male ? creep
   Call: (18) compare(_42940, 3, 4) ? creep
   Exit: (18) compare(<, 3, 4) ? creep
   Exit: (17) compare_offspring(<, prince_andrew, prince_edward) ? creep
   Call: (17) compare_offspring(_51094, prince_charles, prince_andrew) ? creep
   Call: (18) offspring(prince_charles, _51910, _51912) ? creep
   Exit: (18) offspring(prince_charles, male, 1) ? creep
   Call: (18) offspring(prince_andrew, _53542, _53544) ? creep
   Exit: (18) offspring(prince_andrew, male, 3) ? creep
   Call: (18) male=male ? creep
```

SWI-Prolog (AMD64, Multi-threaded, version 9.2.3)

File Edit Settings Run Debug Help

```
   Call: (18) male=male ? creep
   Exit: (18) male=male ? creep
   Call: (18) compare(_51094, 1, 3) ? creep
   Exit: (18) compare(<, 1, 3) ? creep
   Exit: (17) compare_offspring(<, prince_charles, prince_andrew) ? creep
   Call: (19) compare_offspring(_59248, princess_ann, prince_andrew) ? creep
   Call: (20) offspring(princess_ann, _60064, _60066) ? creep
   Exit: (20) offspring(princess_ann, female, 2) ? creep
   Call: (20) offspring(prince_andrew, _61696, _61698) ? creep
   Exit: (20) offspring(prince_andrew, male, 3) ? creep
   Call: (20) female=male ? creep
   Fail: (20) female=male ? creep
   Redo: (19) compare_offspring(_174, princess_ann, prince_andrew) ? creep
   Call: (20) female=male ? creep
   Fail: (20) female=male ? creep
   Redo: (19) compare_offspring(_174, princess_ann, prince_andrew) ? creep
   Call: (20) _174=(>) ? creep
   Exit: (20) (>)=(>) ? creep
   Exit: (19) compare_offspring(>, princess_ann, prince_andrew) ? creep
   Call: (21) compare_offspring(_6574, princess_ann, prince_edward) ? creep
   Call: (22) offspring(princess_ann, _7390, _7392) ? creep
   Exit: (22) offspring(princess_ann, female, 2) ? creep
   Call: (22) offspring(prince_edward, _9022, _9024) ? creep
   Exit: (22) offspring(prince_edward, male, 4) ? creep
   Call: (22) female=male ? creep
   Fail: (22) female=male ? creep
   Redo: (21) compare_offspring(_6574, princess_ann, prince_edward) ? creep
   Call: (22) female=male ? creep
   Fail: (22) female=male ? creep
   Redo: (21) compare_offspring(_6574, princess_ann, prince_edward) ? creep
   Call: (22) _6574=(>) ? creep
   Exit: (22) (>)=(>) ? creep
   Exit: (21) compare_offspring(>, princess_ann, prince_edward) ? creep
^  Exit: (14) sort:predsort(user:compare_offspring, [prince_charles, princess_ann, prince_andrew, prince_edward], [prince_charles, prince_andrew, prince_edw
ard, princess_ann]) ? creep
   Exit: (13) old_line_of_succession([prince_charles, prince_andrew, prince_edward, princess_ann]) ? creep
X = [prince_charles, prince_andrew, prince_edward, princess_ann].

[trace]  ?-
```

Second method:

```
child(charles, elizabeth).
child(ann, elizabeth).
child(andrew, elizabeth).
child(edward, elizabeth).

male(charles).
male(andrew).
male(edward).
female(ann).

birthOrder(charles, 1).
birthOrder(ann, 2).
birthOrder(andrew, 3).
birthOrder(edward, 4).
```

```
?- trace, oldSuccession(X).
   Call: (13) oldSuccession(_47150) ? creep
   Call: (14) male(_47150) ? creep
   Exit: (14) male(charles) ? creep
   Call: (14) child(charles, elizabeth) ? creep
   Exit: (14) child(charles, elizabeth) ? creep
   Call: (14) birthOrder(charles, _51852) ? creep
   Exit: (14) birthOrder(charles, 1) ? creep
   Exit: (13) oldSuccession(charles) ? creep
X = charles ;
   Redo: (14) male(_47150) ? creep
   Exit: (14) male(andrew) ? creep
   Call: (14) child(andrew, elizabeth) ? creep
   Exit: (14) child(andrew, elizabeth) ? creep
   Call: (14) birthOrder(andrew, _59026) ? creep
   Exit: (14) birthOrder(andrew, 3) ? creep
   Exit: (13) oldSuccession(andrew) ? creep
X = andrew ;
   Redo: (14) male(_47150) ? creep
   Exit: (14) male(edward) ? creep
   Call: (14) child(edward, elizabeth) ? creep
   Exit: (14) child(edward, elizabeth) ? creep
   Call: (14) birthOrder(edward, _66200) ? creep
   Exit: (14) birthOrder(edward, 4) ? creep
   Exit: (13) oldSuccession(edward) ? creep
X = edward ;
   Redo: (13) oldSuccession(_47150) ? creep
   Call: (14) female(_47150) ? creep
   Exit: (14) female(ann) ? creep
   Call: (14) child(ann, elizabeth) ? creep
   Exit: (14) child(ann, elizabeth) ? creep
   Call: (14) birthOrder(ann, _74180) ? creep
   Exit: (14) birthOrder(ann, 2) ? creep
   Exit: (13) oldSuccession(ann) ? creep
X = ann.
```

Q2.2

First Method:

```
% Facts about the offspring of Queen Elizabeth
offspring(prince_charles, male, 1).
offspring(princess_ann, female, 2).
offspring(prince_andrew, male, 3).
offspring(prince_edward, male, 4).

% Helper rule to compare offspring for sorting
compare_offspring_new(Order, X, Y) :-
    offspring(X, GenderX, OrderX),
    offspring(Y, GenderY, OrderY),
    compare(Order, OrderX, OrderY).

% Determine the line of succession based on the new rule
new_line_of_succession(SuccessionList) :-
    findall(Name, offspring(Name, _, _), Offspring),
    predsort(compare_offspring_new, Offspring, SuccessionList).
```

```
SWI-Prolog (AMD64, Multi-threaded, version 9.2.3)
File  Edit  Settings  Run  Debug  Help
?- trace, new_line_of_succession(X).
   Call: (13) new_line_of_succession(_12318) ? creep
^  Call: (14) findall(_13736, offspring(_13736, _13742, _13744), _13746) ? creep
   Call: (18) offspring(_13736, _13742, _13744) ? creep
   Exit: (18) offspring(prince_charles, male, 1) ? creep
   Redo: (18) offspring(_13736, _13742, _13744) ? creep
   Exit: (18) offspring(princess_ann, female, 2) ? creep
   Redo: (18) offspring(_13736, _13742, _13744) ? creep
   Exit: (18) offspring(prince_andrew, male, 3) ? creep
   Redo: (18) offspring(_13736, _13742, _13744) ? creep
   Exit: (18) offspring(prince_edward, male, 4) ? creep
^  Exit: (14) findall(_13736, user:offspring(_13736, _13742, _13744), [prince_charles, princess_ann, prince_andrew, prince_edward]) ? creep
^  Call: (14) sort:predsort(compare_offspring_new, [prince_charles, princess_ann, prince_andrew, prince_edward], _12318) ? creep
   Call: (17) compare_offspring_new(_22822, prince_charles, princess_ann) ? creep
   Call: (18) offspring(prince_charles, _23714, _23638) ? creep
   Exit: (18) offspring(prince_charles, male, 1) ? creep
   Call: (18) offspring(princess_ann, _25344, _25268) ? creep
   Exit: (18) offspring(princess_ann, female, 2) ? creep
   Call: (18) compare(_22822, 1, 2) ? creep
   Exit: (18) compare(<, 1, 2) ? creep
   Exit: (17) compare_offspring_new(<, prince_charles, princess_ann) ? creep
   Call: (17) compare_offspring_new(_29356, prince_andrew, prince_edward) ? creep
   Call: (18) offspring(prince_andrew, _30248, _30172) ? creep
   Exit: (18) offspring(prince_andrew, male, 3) ? creep
   Call: (18) offspring(prince_edward, _702, _626) ? creep
   Exit: (18) offspring(prince_edward, male, 4) ? creep
   Call: (18) compare(_156, 3, 4) ? creep
   Exit: (18) compare(<, 3, 4) ? creep
   Exit: (17) compare_offspring_new(<, prince_andrew, prince_edward) ? creep
   Call: (17) compare_offspring_new(_4710, prince_charles, prince_andrew) ? creep
   Call: (18) offspring(prince_charles, _5602, _5526) ? creep
   Exit: (18) offspring(prince_charles, male, 1) ? creep
   Call: (18) offspring(prince_andrew, _7232, _7156) ? creep
   Exit: (18) offspring(prince_andrew, male, 3) ? creep
   Call: (18) compare(_4710, 1, 3) ? creep
   Exit: (18) compare(<, 1, 3) ? creep
   Exit: (17) compare_offspring_new(<, prince_charles, prince_andrew) ? creep
   Call: (19) compare_offspring_new(_11240, princess_ann, prince_andrew) ? creep
   Call: (20) offspring(princess_ann, _12132, _12056) ? creep
   Exit: (20) offspring(princess_ann, female, 2) ? creep
   Call: (20) offspring(prince_andrew, _13762, _13686) ? creep
   Exit: (20) offspring(prince_andrew, male, 3) ? creep
   Call: (20) compare(_11240, 2, 3) ? creep
```

```
SWI-Prolog (AMD64, Multi-threaded, version 9.2.3)
File  Edit  Settings  Run  Debug  Help
   Call: (20) compare(_11240, 2, 3) ? creep
   Exit: (20) compare(<, 2, 3) ? creep
   Exit: (19) compare_offspring_new(<, princess_ann, prince_andrew) ? creep
^  Exit: (14) sort:predsort(user:compare_offspring_new, [prince_charles, princess_ann, prince_andrew, prince_edward], [prince_charles, princess_ann, prince_
andrew, prince_edward]) ? creep
   Exit: (13) new_line_of_succession([prince_charles, princess_ann, prince_andrew, prince_edward]) ? creep
X = [prince_charles, princess_ann, prince_andrew, prince_edward].
```

Second method:

```
child(charles, elizabeth).
child(ann, elizabeth).
child(andrew, elizabeth).
child(edward, elizabeth).

birthOrder(charles, 1).
birthOrder(ann, 2).
birthOrder(andrew, 3).
birthOrder(edward, 4).

newSuccession(X) :-
    child(X, elizabeth),
    birthOrder(X, _).
```

```
?- trace, newSuccession(X).
   Call: (13) newSuccession(_81684) ? creep
   Call: (14) child(_81684, elizabeth) ? creep
   Exit: (14) child(charles, elizabeth) ? creep
   Call: (14) birthOrder(charles, _84774) ? creep
   Exit: (14) birthOrder(charles, 1) ? creep
   Exit: (13) newSuccession(charles) ? creep
X = charles ;
   Redo: (14) child(_81684, elizabeth) ? creep
   Exit: (14) child(ann, elizabeth) ? creep
   Call: (14) birthOrder(ann, _90336) ? creep
   Exit: (14) birthOrder(ann, 2) ? creep
   Exit: (13) newSuccession(ann) ? creep
X = ann ;
   Redo: (14) child(_81684, elizabeth) ? creep
   Exit: (14) child(andrew, elizabeth) ? creep
   Call: (14) birthOrder(andrew, _95898) ? creep
   Exit: (14) birthOrder(andrew, 3) ? creep
   Exit: (13) newSuccession(andrew) ? creep
X = andrew ;
   Redo: (14) child(_81684, elizabeth) ? creep
   Exit: (14) child(edward, elizabeth) ? creep
   Call: (14) birthOrder(edward, _101460) ? creep
   Exit: (14) birthOrder(edward, 4) ? creep
   Exit: (13) newSuccession(edward) ? creep
X = edward.
```