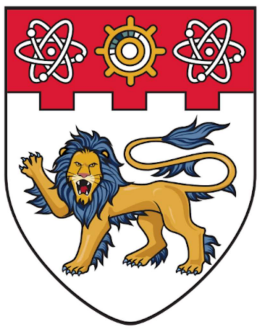

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

Nanyang Technological University
Semester 2, AY23/24

SC3000 - Artificial Intelligence
Lab Assignment 2: Learning to Use Prolog as a Logic Programming Tool

Done By: Lab Group SDAB
Lab Timing: Odd Weeks Tues 0830-1030

Group Name: AStarFound

Liau Zheng Wei U2222032K
Arjun Kumaresh U2221225J
Ronald Teo U2220156E

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 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 succession Logic (FOL).

“sumsum, a competitor of appy”
∀X, ∀Y, company(X) ∧ company(Y) → competitor(X,Y)
In this question, there exists only two companies, so we assume that every company is a competitor of another company. In reality, this is not always the case. So if the question changes and adds another company from another industry, we would have to change this statement.

“A competitor is a rival”
∀X, ∀Y, competitor(X,Y) ⇔ rival(X,Y)
Since they are the same, the two statements are equivalent

“developed some nice smart phone technology”
∃X, ∃T, company(X) ∧ smartphonetechnology(T) → develop(X,T)
In the question, there are two companies but it is not both that developed smartphonetechnology. So given a company with smartphonetechnology, that company may have developed it as the company can also use smartphonetechnology that it did not develop .

“Smartphone technology is a business”
∀T, smartphonetechnology(T) ⇔ business(T)
smartphonetechnology is equivalent to a business

“all of which was stolen by stevey”
∃P, ∃T, person(P) ∧ smartphonetechnology(T) → steals(P,T)
A person can steal smartphonetechnology

“stevey, who is a boss of appy”
∀X, ∃P, person(P) ∧ company(X) → boss(P,X)
For every company, there is a person who is the boss there. We also can determine that stevey is a person who is the boss of appy

“It is unethical for a boss to steal business from rival companies”
In this statement, it explains that it is unethical when certain conditions are met. This includes:

1. boss P is from company X is boss(P,X)
2. business is stolen by boss P: steals(P,T)
3. Two rival companies exist: rival(X,Y)

4. company Y developed the technology which was stolen: develop(Y,T)

Together, they form the statement:

$\exists P, \exists T, \exists X, \exists Y,$ boss(P,X) $\wedge$ steals(P,T) $\wedge$ rival(X,Y) $\wedge$ develop(Y,T) $\rightarrow \neg$ethical(P)

2. Write these FOL statements as Prolog clauses.

```
:- discontiguous smartphonetechnology/1.

% Facts
% Both appy and sumsum are companies
company(appy).
company(sumsum).

% stevey is a person
person(stevey).

% galactica-s3 is a smartphone technology
smartphonetechnology(galactica_s3).

% sumsum developed galactica-s3
develop(sumsum,galactica_s3) :- company(sumsum), smartphonetechnology(galactica_s3).

% stevey stole galactica-s3
steals(stevey,galactica_s3) :- person(stevey), smartphonetechnology(galactica_s3).

% stevey is the boss of appy
boss(stevey,appy) :- company(appy), person(stevey).

% Inferences
% There are two competing companies
competitor(X,Y) :- company(X), company(Y).

% A rival is a competitor
competitor(X,Y) :- rival(X,Y).
rival(X,Y) :- competitor(X,Y).

% A smartphonetechnology is a business
smartphonetechnology(T):- business(T).
business(T) :- smartphonetechnology(T).

% It is unethical for a boss of a company to steal business from rival companies
unethical(P) :- boss(P,X), steals(P,T), rival(X,Y), develop(Y,T).
```

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

```
% c:/Users/liauz/OneDrive/Desktop/AIAssignmentEx2.pl compiled 0.00 sec, 13 clauses
?- unethical(X).
X = stevey .

?- trace.
true.

[trace]  ?- unethical(X).
   Call: (12) unethical(_16942) ? creep
   Call: (13) boss(_16942, _18228) ? creep
   Call: (14) company(appy) ? creep
   Exit: (14) company(appy) ? creep
   Call: (14) person(stevey) ? creep
   Exit: (14) person(stevey) ? creep
   Exit: (13) boss(stevey, appy) ? creep
   Call: (13) steals(stevey, _23074) ? creep
   Call: (14) person(stevey) ? creep
   Exit: (14) person(stevey) ? creep
   Call: (14) smartphonetechnology(galactica_s3) ? creep
   Exit: (14) smartphonetechnology(galactica_s3) ? creep
   Exit: (13) steals(stevey, galactica_s3) ? creep
   Call: (13) rival(appy, _27920) ? creep
   Call: (14) competitor(appy, _27920) ? creep
   Call: (15) company(appy) ? creep
   Exit: (15) company(appy) ? creep
   Call: (15) company(_27920) ? creep
   Exit: (15) company(appy) ? creep
   Exit: (14) competitor(appy, appy) ? creep
   Exit: (13) rival(appy, appy) ? creep
   Call: (13) develop(appy, galactica_s3) ? creep
   Fail: (13) develop(appy, galactica_s3) ? creep
   Redo: (15) company(_27920) ? creep
   Exit: (15) company(sumsum) ? creep
   Exit: (14) competitor(appy, sumsum) ? creep
   Exit: (13) rival(appy, sumsum) ? creep
   Call: (13) develop(sumsum, galactica_s3) ? creep
   Call: (14) company(sumsum) ? creep
   Exit: (14) company(sumsum) ? creep
   Call: (14) smartphonetechnology(galactica_s3) ? creep
   Exit: (14) smartphonetechnology(galactica_s3) ? creep
   Exit: (13) develop(sumsum, galactica_s3) ? creep
   Exit: (12) unethical(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.

2.1 . 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.

```
% Gender Definition
male(prince_charles).
male(prince_andrew).
male(prince_edward).
female(princess_ann).
female(queen_elizabeth).

% Define monarch
monarch(queen_elizabeth).

% Define parent-child relation
parent(queen_elizabeth, prince_charles).
parent(queen_elizabeth, prince_andrew).
parent(queen_elizabeth, prince_edward).
parent(queen_elizabeth, princess_ann).

% Define birth order
older(queen_elizabeth, prince_charles).
older(prince_charles, princess_ann).
older(princess_ann, prince_andrew).
older(prince_andrew, prince_edward).

% Determine birth older transitivity
older_than_x(A,B) :- older(A,B).
older_than_x(A,C) :-
        older(A,B), older_than_x(B,C).

% Checks if same gender
```

```
same_gender(X,Y) :- ((male(X), male(Y)); (female(X), female(Y))), not(X=Y).

% Check if different gender
different_gender(X,Y) :- male(X), female(Y).

% Checks if same gender and X is older than Y.
same_gender_older(X,Y) :- same_gender(X,Y), older_than_x(X,Y).

% Helper rule to verify correct succession order between two individuals
correct_order(X, Y) :-
    (same_gender_older(X, Y) ; different_gender(X, Y)),
    \+monarch(X), \+monarch(Y).

% Validates the order of a list of royals
valid_order([]).
valid_order([_]).
valid_order([First,Second|Rest]) :-
    correct_order(First, Second),
    valid_order([Second|Rest]).

% Main method to get the valid order of a specific number of royals
get_royal_order(List, Number) :-
    length(List, Number),
    valid_order(List).
```

Definitions:

Gender Definitions and Monarch Definition

Defining the gender for the prince and princess and, defining the monarch - queen elizabeth. This is crucial for determining the order of succession later on.

Parent-Child Relations

This defines that parent-child relationship between queen and the princes and princesses. This definition is not mandatory for solving the problem, but it makes it clear who are the parent and child.

Birth Order Definitions

This defines the order of birth.

Older Than Relation (Transitivity)

This predicate help to check for transitivity relation whereby If A is older than B, B is older than C, then A is older than C.

Same Gender Check

A predicate that determines if two individuals are of the same gender.

Same Gender and Older Check
A predicate that checks if two individuals - X and Y and the same gender and, whether X is older than Y.

Different Gender Check
A predicate that determines if two individuals are different gender.

Correct Order Check
This predicate checks if the order of succession between two individual is correct. It takes into account the gender and age conditions specified in the question and ensuring that none of them are monarchs.

Valid Order of Royals
The valid_order predicate recursively checks a list of individuals to ensure each pair is in the correct succession order.

Main Method for Valid Order
The get_royal_order predicate validate a list of royal family members against the conditions to determine if they are in a correct order of succession.

Trace

```
[trace]  ?- get_royal_order(X, 4).
   Call: (12) get_royal_order(_22326, 4) ? creep
   Call: (13) length(_22326, 4) ? creep
   Exit: (13) length([_24440, _24446, _24452, _24458], 4) ? creep
   Call: (13) valid_order([_24440, _24446, _24452, _24458]) ? creep
   Call: (14) correct_order(_24440, _24446) ? creep
   Call: (15) same_gender_older(_24440, _24446) ? creep
   Call: (16) same_gender(_24440, _24446) ? creep
   Call: (17) male(_24440) ? creep
   Exit: (17) male(prince_charles) ? creep
   Call: (17) male(_24446) ? creep
   Exit: (17) male(prince_charles) ? creep
^  Call: (17) not(prince_charles=prince_charles) ? creep
^  Fail: (17) not(user:(prince_charles=prince_charles)) ? creep
   Redo: (17) male(_24446) ? creep
   Exit: (17) male(prince_andrew) ? creep
^  Call: (17) not(prince_charles=prince_andrew) ? creep
^  Exit: (17) not(user:(prince_charles=prince_andrew)) ? creep
   Exit: (16) same_gender(prince_charles, prince_andrew) ? creep
   Call: (16) older_than_x(prince_charles, prince_andrew) ? creep
   Call: (17) older(prince_charles, prince_andrew) ? creep
   Fail: (17) older(prince_charles, prince_andrew) ? creep
   Redo: (16) older_than_x(prince_charles, prince_andrew) ? creep
   Call: (17) older(prince_charles, _40690) ? creep
   Exit: (17) older(prince_charles, princess_ann) ? creep
   Call: (17) older_than_x(princess_ann, prince_andrew) ? creep
   Call: (18) older(princess_ann, prince_andrew) ? creep
   Exit: (18) older(princess_ann, prince_andrew) ? creep
   Exit: (17) older_than_x(princess_ann, prince_andrew) ? creep
   Exit: (16) older_than_x(prince_charles, prince_andrew) ? creep
   Exit: (15) same_gender_older(prince_charles, prince_andrew) ? creep
   Call: (15) monarch(prince_charles) ? creep
   Fail: (15) monarch(prince_charles) ? creep
   Redo: (14) correct_order(prince_charles, prince_andrew) ? creep
   Call: (15) monarch(prince_andrew) ? creep
   Fail: (15) monarch(prince_andrew) ? creep
   Redo: (14) correct_order(prince_charles, prince_andrew) ? creep
   Exit: (14) correct_order(prince_charles, prince_andrew) ? creep
   Call: (14) valid_order([prince_andrew, _24452, _24458]) ? creep
   Call: (15) correct_order(prince_andrew, _24452) ? creep
   Call: (16) same_gender_older(prince_andrew, _24452) ? creep
   Call: (17) same_gender(prince_andrew, _24452) ? creep
   Call: (18) male(prince_andrew) ? creep
   Exit: (18) male(prince_andrew) ? creep
   Call: (18) male(_24452) ? creep
   Exit: (18) male(prince_charles) ? creep
^  Call: (18) not(prince_andrew=prince_charles) ? creep
^  Exit: (18) not(user:(prince_andrew=prince_charles)) ? creep
   Exit: (17) same_gender(prince_andrew, prince_charles) ? creep
   Call: (17) older_than_x(prince_andrew, prince_charles) ? creep
   Call: (18) older(prince_andrew, prince_charles) ? creep
   Fail: (18) older(prince_andrew, prince_charles) ? creep
   Redo: (17) older_than_x(prince_andrew, prince_charles) ? creep
   Call: (18) older(prince_andrew, _922) ? creep
   Exit: (18) older(prince_andrew, prince_edward) ? creep
   Call: (18) older_than_x(prince_edward, prince_charles) ? creep
   Call: (19) older(prince_edward, prince_charles) ? creep
   Fail: (19) older(prince_edward, prince_charles) ? creep
   Redo: (18) older_than_x(prince_edward, prince_charles) ? creep
   Call: (19) older(prince_edward, _5784) ? creep
   Fail: (19) older(prince_edward, _5784) ? creep
   Fail: (18) older_than_x(prince_edward, prince_charles) ? creep
   Fail: (17) older_than_x(prince_andrew, prince_charles) ? creep
```

```
   Redo: (18) male(_130) ? creep
   Exit: (18) male(prince_andrew) ? creep
^  Call: (18) not(prince_andrew=prince_andrew) ? creep
^  Fail: (18) not(user:(prince_andrew=prince_andrew)) ? creep
   Redo: (18) male(_130) ? creep
   Exit: (18) male(prince_edward) ? creep
^  Call: (18) not(prince_andrew=prince_edward) ? creep
^  Exit: (18) not(user:(prince_andrew=prince_edward)) ? creep
   Exit: (17) same_gender(prince_andrew, prince_edward) ? creep
   Call: (17) older_than_x(prince_andrew, prince_edward) ? creep
   Call: (18) older(prince_andrew, prince_edward) ? creep
   Exit: (18) older(prince_andrew, prince_edward) ? creep
   Exit: (17) older_than_x(prince_andrew, prince_edward) ? creep
   Exit: (16) same_gender_older(prince_andrew, prince_edward) ? creep
   Call: (16) monarch(prince_andrew) ? creep
   Fail: (16) monarch(prince_andrew) ? creep
   Redo: (15) correct_order(prince_andrew, prince_edward) ? creep
   Call: (16) monarch(prince_edward) ? creep
   Fail: (16) monarch(prince_edward) ? creep
   Redo: (15) correct_order(prince_andrew, prince_edward) ? creep
   Exit: (15) correct_order(prince_andrew, prince_edward) ? creep
   Call: (15) valid_order([prince_edward, _136]) ? creep
   Call: (16) correct_order(prince_edward, _136) ? creep
   Call: (17) same_gender_older(prince_edward, _136) ? creep
   Call: (18) same_gender(prince_edward, _136) ? creep
   Call: (19) male(prince_edward) ? creep
   Exit: (19) male(prince_edward) ? creep
   Call: (19) male(_136) ? creep
   Exit: (19) male(prince_charles) ? creep
^  Call: (19) not(prince_edward=prince_charles) ? creep
^  Exit: (19) not(user:(prince_edward=prince_charles)) ? creep
   Exit: (18) same_gender(prince_edward, prince_charles) ? creep
   Call: (18) older_than_x(prince_edward, prince_charles) ? creep
   Call: (19) older(prince_edward, prince_charles) ? creep
   Fail: (19) older(prince_edward, prince_charles) ? creep
   Redo: (18) older_than_x(prince_edward, prince_charles) ? creep
   Call: (19) older(prince_edward, _38224) ? creep
   Fail: (19) older(prince_edward, _38224) ? creep
   Fail: (18) older_than_x(prince_edward, prince_charles) ? creep
   Redo: (19) male(_136) ? creep
   Exit: (19) male(prince_andrew) ? creep
^  Call: (19) not(prince_edward=prince_andrew) ? creep
^  Exit: (19) not(user:(prince_edward=prince_andrew)) ? creep
   Exit: (18) same_gender(prince_edward, prince_andrew) ? creep
   Call: (18) older_than_x(prince_edward, prince_andrew) ? creep
   Call: (19) older(prince_edward, prince_andrew) ? creep
   Fail: (19) older(prince_edward, prince_andrew) ? creep
   Redo: (18) older_than_x(prince_edward, prince_andrew) ? creep
   Call: (19) older(prince_edward, _47966) ? creep
   Fail: (19) older(prince_edward, _47966) ? creep
   Fail: (18) older_than_x(prince_edward, prince_andrew) ? creep
   Redo: (19) male(_136) ? creep
   Exit: (19) male(prince_edward) ? creep
^  Call: (19) not(prince_edward=prince_edward) ? creep
^  Fail: (19) not(user:(prince_edward=prince_edward)) ? creep
   Redo: (18) same_gender(prince_edward, _136) ? creep
   Call: (19) female(prince_edward) ? creep
   Fail: (19) female(prince_edward) ? creep
   Fail: (18) same_gender(prince_edward, _136) ? creep
   Fail: (17) same_gender_older(prince_edward, _136) ? creep
   Redo: (16) correct_order(prince_edward, _136) ? creep
```

```
   Call: (17) different_gender(prince_edward, _136) ? creep
   Call: (18) male(prince_edward) ? creep
   Exit: (18) male(prince_edward) ? creep
   Call: (18) female(_136) ? creep
   Exit: (18) female(princess_ann) ? creep
   Exit: (17) different_gender(prince_edward, princess_ann) ? creep
   Call: (17) monarch(prince_edward) ? creep
   Fail: (17) monarch(prince_edward) ? creep
   Redo: (16) correct_order(prince_edward, princess_ann) ? creep
   Call: (17) monarch(princess_ann) ? creep
   Fail: (17) monarch(princess_ann) ? creep
   Redo: (16) correct_order(prince_edward, princess_ann) ? creep
   Exit: (16) correct_order(prince_edward, princess_ann) ? creep
   Call: (16) valid_order([princess_ann]) ? creep
   Exit: (16) valid_order([princess_ann]) ? creep
   Exit: (15) valid_order([prince_edward, princess_ann]) ? creep
   Exit: (14) valid_order([prince_andrew, prince_edward, princess_ann]) ? creep
   Exit: (13) valid_order([prince_charles, prince_andrew, prince_edward, princess_ann]) ?
 creep
   Exit: (12) get_royal_order([prince_charles, prince_andrew, prince_edward, princess_ann
], 4) ? creep
X = [prince_charles, prince_andrew, prince_edward, princess_ann] .
```

Logic Flow:

1. Defining the attributes, relations and predicates,
2. We check is two person - A and B are the same gender.
   2.1. If they are, then A must be older than B. This means that A will be placed before B in the succession order.
   2.2. If they are not, then A is a male and B is female. This means that A will be placed before B in the succession order.
3. We recursively check the list of potential successors until the list is right.

2.2 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

```
% Gender Definitions - Kept for completeness.
male(prince_charles).
male(prince_andrew).
male(prince_edward).
female(princess_ann).
female(queen_elizabeth).

% Monarch Definition
monarch(queen_elizabeth).

% Parent-Child Relations
parent(queen_elizabeth, prince_charles).
parent(queen_elizabeth, prince_andrew).
parent(queen_elizabeth, prince_edward).
parent(queen_elizabeth, princess_ann).

% Birth Order Definitions
older(prince_charles, princess_ann).
older(princess_ann, prince_andrew).
older(prince_andrew, prince_edward).

% Determine if A is older than C using transitivity
older_than(A, C) :- older(A, C).
older_than(A, C) :- older(A, B), older_than(B, C).

% Validates the order of a list of royals based on birth order alone
valid_order([]).
valid_order([_]).
valid_order([First, Second|Rest]) :-
    older_than(First, Second),
    valid_order([Second|Rest]).

% Main method to get the valid order of a specific number of royals
get_royal_order(List, Number) :-
    length(List, Number),
    valid_order(List).
```

Trace:

```
[trace]  ?- get_royal_order(X, 4).
   Call: (12) get_royal_order(_22326, 4) ? creep
   Call: (13) length(_22326, 4) ? creep
   Exit: (13) length([_24440, _24446, _24452, _24458], 4) ? creep
   Call: (13) valid_order([_24440, _24446, _24452, _24458]) ? creep
   Call: (14) older_than(_24440, _24446) ? creep
   Call: (15) older(_24440, _24446) ? creep
   Exit: (15) older(prince_charles, princess_ann) ? creep
   Exit: (14) older_than(prince_charles, princess_ann) ? creep
   Call: (14) valid_order([princess_ann, _24452, _24458]) ? creep
   Call: (15) older_than(princess_ann, _24452) ? creep
   Call: (16) older(princess_ann, _24452) ? creep
   Exit: (16) older(princess_ann, prince_andrew) ? creep
   Exit: (15) older_than(princess_ann, prince_andrew) ? creep
   Call: (15) valid_order([prince_andrew, _24458]) ? creep
   Call: (16) older_than(prince_andrew, _24458) ? creep
   Call: (17) older(prince_andrew, _24458) ? creep
   Exit: (17) older(prince_andrew, prince_edward) ? creep
   Exit: (16) older_than(prince_andrew, prince_edward) ? creep
   Call: (16) valid_order([prince_edward]) ? creep
   Exit: (16) valid_order([prince_edward]) ? creep
   Exit: (15) valid_order([prince_andrew, prince_edward]) ? creep
   Exit: (14) valid_order([princess_ann, prince_andrew, prince_edward]) ? creep
   Exit: (13) valid_order([prince_charles, princess_ann, prince_andrew, prince_edward]) ?
 creep
   Exit: (12) get_royal_order([prince_charles, princess_ann, prince_andrew, prince_edward
], 4) ? creep
X = [prince_charles, princess_ann, prince_andrew, prince_edward] .
```

Logic Process Summary:

1. The definition of attributes, relationship and predicate are mostly similar to those in part 2.1. However, some predicate are removes and amended. We now no longer perform gender checks - correct_order predicate. We also modified the valid_order predicate by replacing correct_order with older_than.
2. Similar to part 2.1, we recursively check the list of potential successors until the list is right.