Zhang Yuhan, U1823060F, SSP2

Exercise 1: 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. It is unethical for a Boss to steal business from rival companies. A competitor of Appy is a rival. Smart phone technology is a business.

In order to prove that Stevey is unethical, smart phone technology must be a business, SumSum and Appy must be competitors, and Stevey, the boss of Appy, must steal the smart phone technology (Galactica-S3) from SumSum. This is represented by the prolog code below.

```
company(sumsum).
company(appy).
developed(sumsum,galactica-s3).
boss(stevey).
steal(stevey,galactica-s3,sumsum).
technology(galactica-s3).
competitor(sumsum,appy).


business(X,Y):- technology(X),developed(Y,X).
rival(X):- competitor(X,appy);competitor(appy,X).
unethical(X):- boss(X),business(Y,Z),rival(Z),steal(X,Y,Z).
```

The trace result of unethical(stevey). Is shown below. Since copy and pasting from the terminal caused format issues, the trace results in the txt files for both programs are generated from SWISH instead.

```
3 ?- trace,unethical(stevey).
   Call: (9) unethical(stevey) ? creep
   Call: (10) boss(stevey) ? creep
   Exit: (10) boss(stevey) ? creep
   Call: (10) business(_4056, _4058) ? creep
   Call: (11) technology(_4056) ? creep
   Exit: (11) technology(galactica-s3) ? creep
   Call: (11) developed(_4062, galactica-s3) ? creep
   Exit: (11) developed(sumsum, galactica-s3) ? creep
   Exit: (10) business(galactica-s3, sumsum) ? creep
   Call: (10) rival(sumsum) ? creep
   Call: (11) competitor(sumsum, appy) ? creep
   Exit: (11) competitor(sumsum, appy) ? creep
   Exit: (10) rival(sumsum) ? creep
   Call: (10) steal(stevey, galactica-s3, sumsum) ? creep
   Exit: (10) steal(stevey, galactica-s3, sumsum) ? creep
   Exit: (9) unethical(stevey) ? creep
true ;
   Redo: (10) rival(sumsum) ? creep
   Call: (11) competitor(appy, sumsum) ? creep
   Fail: (11) competitor(appy, sumsum) ? creep
   Fail: (10) rival(sumsum) ? creep
   Fail: (9) unethical(stevey) ? creep
false.
```

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.
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.

For the old succession rule, gender takes priority over age. This means that a prince will always have priority over a princess regardless of their age. Age is considered only if there are multiple princes or there are multiple princesses and no princes. This can be shown in the precedes_old(X,Y) function in the screenshot below. Here is the breakdown of the code:

- If X and Y are both princes, return true if X is older than Y. Else if X is a prince and Y is a princess, then return true if both conditions are met. Else if both X and Y are princesses, return true if X is older than Y.

For the new succession rule, age is the only determining factor. This means that gender no longer plays a role in the order of succession. This can be shown in the precedes_new(X,Y) function in the screenshot below. Here is the breakdown of the code:

- If X and Y are both princes, return true if X is older than Y. Else if X is a prince and Y is a princess, return true if X is older than Y. Else if X is a princess and Y is a prince, return true if X is older than Y. Else if X and Y are princesses, return true if X is older than Y.

The insert and succession_order functions utilize the precedes rules to sort the successor candidates in the order of first to last. Succession_order recursively separates the first value from the offspring_list and uses a temporary list to store the rest of the values. Then, the insert function is called to insert each value into the succession_line list in the correct order.

The succession_list function is responsible for finding all offspring of a monarch and passing the offspring_list to the succession_order function.

The prolog code for Royal family succession is shown below:

```
queen(elizabeth).
prince(charles).
prince(andrew).
prince(edward).
princess(ann).

offspring_of(charles,elizabeth).
offspring_of(andrew,elizabeth).
offspring_of(edward,elizabeth).
offspring_of(ann,elizabeth).

older_than(charles,ann).
older_than(ann,andrew).
older_than(andrew,edward).

is_older(X,Y) :- older_than(X,Y);(older_than(X,_),older_than(_,Y)).

/*Old Succession order*/
/*Gender has higher priority than age for order of succession, thus Princes have priority over princesses.*/
/*precedes function makes sure that the princes have higher priority before checking age.*/

precedes_old(X,Y) :- prince(X),prince(Y),is_older(X,Y);
    prince(X),princess(Y);
    princess(X),princess(Y),is_older(X,Y).

insert_old(A, [B|C], [B|D]):- (precedes_old(B, A)), !, insert_old(A, C, D).
insert_old(A, C, [A|C]).
succession_order_old([A|B], Sorted_List):- succession_order_old(B, Partial_order),
    insert_old(A, Partial_order, Sorted_List).
succession_order_old([], []).

succession_list_old(Queen, Succession_line):-
    findall(Y, offspring_of(Y, Queen),Offspring_list),
    succession_order_old(Offspring_list, Succession_line).

/*New succession order*/
/*Gender is does not affect priority of succession, thus age is the only determining factor.*/
/*precedes function now only takes age into account.*/

precedes_new(X,Y) :- prince(X),prince(Y),is_older(X,Y);
    prince(X),princess(Y),is_older(X,Y);
    princess(X),prince(Y),is_older(X,Y);
    princess(X),princess(Y),is_older(X,Y).

insert_new(A, [B|C], [B|D]):- (precedes_new(B, A)), !, insert_new(A, C, D).
insert_new(A, C, [A|C]).

succession_order_new([A|B], Sorted_List):- succession_order_new(B, Partial_order),
    insert_new(A, Partial_order, Sorted_List).
succession_order_new([], []).

succession_list_new(Queen, Succession_line):-
    findall(Y, offspring_of(Y, Queen),Offspring_list),
    succession_order_new(Offspring_list, Succession_line).
```

The trace result of succession_list_old(elizabeth, Succession_line) is shown below. Since copy and pasting from the terminal caused format issues, the trace results in the txt files for both programs are generated from SWISH instead.

```
3 ?- trace,succession_list_old(elizabeth,Succession_line).
   Call: (9) succession_list_old(elizabeth, _4550) ? creep
^  Call: (10) findall(_4854, offspring_of(_4854, elizabeth), _4878) ? creep
   Call: (15) offspring_of(_4854, elizabeth) ? creep
   Exit: (15) offspring_of(charles, elizabeth) ? creep
   Redo: (15) offspring_of(_4854, elizabeth) ? creep
   Exit: (15) offspring_of(andrew, elizabeth) ? creep
   Redo: (15) offspring_of(_4854, elizabeth) ? creep
   Exit: (15) offspring_of(edward, elizabeth) ? creep
   Redo: (15) offspring_of(_4854, elizabeth) ? creep
   Exit: (15) offspring_of(ann, elizabeth) ? creep
^  Call: (15) call('$bags':'$destroy_findall_bag') ? creep
^  Exit: (15) call('$bags':'$destroy_findall_bag') ? creep
^  Exit: (10) findall(_4854, user:offspring_of(_4854, elizabeth), [charles, andrew, edward, ann]) ? creep
   Call: (10) succession_order_old([charles, andrew, edward, ann], _4550) ? creep
   Call: (11) succession_order_old([andrew, edward, ann], _4934) ? creep
   Call: (12) succession_order_old([edward, ann], _4934) ? creep
   Call: (13) succession_order_old([ann], _4934) ? creep
   Call: (14) succession_order_old([], _4934) ? creep
   Exit: (14) succession_order_old([], []) ? creep
   Call: (14) insert_old(ann, [], _4936) ? creep
   Exit: (14) insert_old(ann, [], [ann]) ? creep
   Exit: (13) succession_order_old([ann], [ann]) ? creep
   Call: (13) insert_old(edward, [ann], _4942) ? creep
   Call: (14) precedes_old(ann, edward) ? creep
   Call: (15) prince(ann) ? creep
   Fail: (15) prince(ann) ? creep
   Redo: (14) precedes_old(ann, edward) ? creep
   Call: (15) prince(ann) ? creep
   Fail: (15) prince(ann) ? creep
   Redo: (14) precedes_old(ann, edward) ? creep
   Call: (15) princess(ann) ? creep
   Exit: (15) princess(ann) ? creep
   Call: (15) princess(edward) ? creep
   Fail: (15) princess(edward) ? creep
   Fail: (14) precedes_old(ann, edward) ? creep
   Redo: (13) insert_old(edward, [ann], _4942) ? creep
   Exit: (13) insert_old(edward, [ann], [edward, ann]) ? creep
   Exit: (12) succession_order_old([edward, ann], [edward, ann]) ? creep
   Call: (12) insert_old(andrew, [edward, ann], _4948) ? creep
   Call: (13) precedes_old(edward, andrew) ? creep
   Call: (14) prince(edward) ? creep
   Exit: (14) prince(edward) ? creep
   Call: (14) prince(andrew) ? creep
   Exit: (14) prince(andrew) ? creep
   Call: (14) is_older(edward, andrew) ? creep
   Call: (15) older_than(edward, andrew) ? creep
   Fail: (15) older_than(edward, andrew) ? creep
   Redo: (14) is_older(edward, andrew) ? creep
   Call: (15) older_than(edward, _4952) ? creep
   Fail: (15) older_than(edward, _4952) ? creep
   Fail: (14) is_older(edward, andrew) ? creep
   Redo: (13) precedes_old(edward, andrew) ? creep
   Call: (14) prince(edward) ? creep
   Exit: (14) prince(edward) ? creep
   Call: (14) princess(andrew) ? creep
   Fail: (14) princess(andrew) ? creep
   Redo: (13) precedes_old(edward, andrew) ? creep
   Call: (14) princess(edward) ? creep
   Fail: (14) princess(edward) ? creep
   Fail: (13) precedes_old(edward, andrew) ? creep
   Redo: (12) insert_old(andrew, [edward, ann], _4948) ? creep
   Exit: (12) insert_old(andrew, [edward, ann], [andrew, edward, ann]) ? creep
   Exit: (11) succession_order_old([andrew, edward, ann], [andrew, edward, ann]) ? creep
   Call: (11) insert_old(charles, [andrew, edward, ann], _4550) ? creep
   Call: (12) precedes_old(andrew, charles) ? creep
   Call: (13) prince(andrew) ? creep
   Exit: (13) prince(andrew) ? creep
   Call: (13) prince(charles) ? creep
   Exit: (13) prince(charles) ? creep
   Call: (13) is_older(andrew, charles) ? creep
```

```
   Call: (14) older_than(andrew, charles) ? creep
   Fail: (14) older_than(andrew, charles) ? creep
   Redo: (13) is_older(andrew, charles) ? creep
   Call: (14) older_than(andrew, _4958) ? creep
   Exit: (14) older_than(andrew, edward) ? creep
   Call: (14) older_than(_4956, charles) ? creep
   Fail: (14) older_than(_4956, charles) ? creep
   Fail: (13) is_older(andrew, charles) ? creep
   Redo: (12) precedes_old(andrew, charles) ? creep
   Call: (13) prince(andrew) ? creep
   Exit: (13) prince(andrew) ? creep
   Call: (13) princess(charles) ? creep
   Fail: (13) princess(charles) ? creep
   Redo: (12) precedes_old(andrew, charles) ? creep
   Call: (13) princess(andrew) ? creep
   Fail: (13) princess(andrew) ? creep
   Fail: (12) precedes_old(andrew, charles) ? creep
   Redo: (11) insert_old(charles, [andrew, edward, ann], _4550) ? creep
   Exit: (11) insert_old(charles, [andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (10) succession_order_old([charles, andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (9) succession_list_old(elizabeth, [charles, andrew, edward, ann]) ? creep
Succession_line = [charles, andrew, edward, ann].
```

The trace result of succession_list_new(elizabeth, Succession_line) is shown below.

```
trace,succession_list_new(elizabeth,Succession_line).
   Call: (9) succession_list_new(elizabeth, _6020) ? creep
^  Call: (10) findall(_6320, offspring_of(_6320, elizabeth), _6344) ? creep
   Call: (15) offspring_of(_6320, elizabeth) ? creep
   Exit: (15) offspring_of(charles, elizabeth) ? creep
   Redo: (15) offspring_of(_6320, elizabeth) ? creep
   Exit: (15) offspring_of(andrew, elizabeth) ? creep
   Redo: (15) offspring_of(_6320, elizabeth) ? creep
   Exit: (15) offspring_of(edward, elizabeth) ? creep
   Redo: (15) offspring_of(_6320, elizabeth) ? creep
   Exit: (15) offspring_of(ann, elizabeth) ? creep
^  Call: (15) call('$bags':'$destroy_findall_bag') ? creep
^  Exit: (15) call('$bags':'$destroy_findall_bag') ? creep
^  Exit: (10) findall(_6320, user:offspring_of(_6320, elizabeth), [charles, andrew, edward, ann]) ? creep
   Call: (10) succession_order_new([charles, andrew, edward, ann], _6020) ? creep
   Call: (11) succession_order_new([andrew, edward, ann], _6400) ? creep
   Call: (12) succession_order_new([edward, ann], _6400) ? creep
   Call: (13) succession_order_new([ann], _6400) ? creep
   Call: (14) succession_order_new([], _6400) ? creep
   Exit: (14) succession_order_new([], []) ? creep
   Call: (14) insert_new(ann, [], _6402) ? creep
   Exit: (14) insert_new(ann, [], [ann]) ? creep
   Exit: (13) succession_order_new([ann], [ann]) ? creep
   Call: (13) insert_new(edward, [ann], _6408) ? creep
   Call: (14) precedes_new(ann, edward) ? creep
   Call: (15) prince(ann) ? creep
   Fail: (15) prince(ann) ? creep
   Redo: (14) precedes_new(ann, edward) ? creep
   Call: (15) prince(ann) ? creep
   Fail: (15) prince(ann) ? creep
   Redo: (14) precedes_new(ann, edward) ? creep
   Call: (15) princess(ann) ? creep
   Exit: (15) princess(ann) ? creep
   Call: (15) prince(edward) ? creep
   Exit: (15) prince(edward) ? creep
   Call: (15) is_older(ann, edward) ? creep
   Call: (16) older_than(ann, edward) ? creep
   Fail: (16) older_than(ann, edward) ? creep
   Redo: (15) is_older(ann, edward) ? creep
   Call: (16) older_than(ann, _6412) ? creep
   Exit: (16) older_than(ann, andrew) ? creep
   Call: (16) older_than(_6410, edward) ? creep
   Exit: (16) older_than(andrew, edward) ? creep
   Exit: (15) is_older(ann, edward) ? creep
   Exit: (14) precedes_new(ann, edward) ? creep
   Call: (14) insert_new(edward, [], _6392) ? creep
```

```
Exit: (14) insert_new(edward, [], [edward]) ? creep
Exit: (13) insert_new(edward, [ann], [ann, edward]) ? creep
Exit: (12) succession_order_new([edward, ann], [ann, edward]) ? creep
Call: (12) insert_new(andrew, [ann, edward], _6420) ? creep
Call: (13) precedes_new(ann, andrew) ? creep
Call: (14) prince(ann) ? creep
Fail: (14) prince(ann) ? creep
Redo: (13) precedes_new(ann, andrew) ? creep
Call: (14) prince(ann) ? creep
Fail: (14) prince(ann) ? creep
Redo: (13) precedes_new(ann, andrew) ? creep
Call: (14) princess(ann) ? creep
Exit: (14) princess(ann) ? creep
Call: (14) prince(andrew) ? creep
Exit: (14) prince(andrew) ? creep
Call: (14) is_older(ann, andrew) ? creep
Call: (15) older_than(ann, andrew) ? creep
Exit: (15) older_than(ann, andrew) ? creep
Exit: (14) is_older(ann, andrew) ? creep
Exit: (13) precedes_new(ann, andrew) ? creep
Call: (13) insert_new(andrew, [edward], _6404) ? creep
Call: (14) precedes_new(edward, andrew) ? creep
Call: (15) prince(edward) ? creep
Exit: (15) prince(edward) ? creep
Call: (15) prince(andrew) ? creep
Exit: (15) prince(andrew) ? creep
Call: (15) is_older(edward, andrew) ? creep
Call: (16) older_than(edward, andrew) ? creep
Fail: (16) older_than(edward, andrew) ? creep
Redo: (15) is_older(edward, andrew) ? creep
Call: (16) older_than(edward, _6430) ? creep
Fail: (16) older_than(edward, _6430) ? creep
Fail: (15) is_older(edward, andrew) ? creep
Redo: (14) precedes_new(edward, andrew) ? creep
Call: (15) prince(edward) ? creep
Exit: (15) prince(edward) ? creep
Call: (15) princess(andrew) ? creep
Fail: (15) princess(andrew) ? creep
Redo: (14) precedes_new(edward, andrew) ? creep
Call: (15) princess(edward) ? creep
Fail: (15) princess(edward) ? creep
Redo: (14) precedes_new(edward, andrew) ? creep
Call: (15) princess(edward) ? creep
Fail: (15) princess(edward) ? creep
Fail: (14) precedes_new(edward, andrew) ? creep
Redo: (13) insert_new(andrew, [edward], _6404) ? creep
Exit: (13) insert_new(andrew, [edward], [andrew, edward]) ? creep
Exit: (12) insert_new(andrew, [ann, edward], [ann, andrew, edward]) ? creep
Exit: (11) succession_order_new([andrew, edward, ann], [ann, andrew, edward]) ? creep
Call: (11) insert_new(charles, [ann, andrew, edward], _6020) ? creep
Call: (12) precedes_new(ann, charles) ? creep
Call: (13) prince(ann) ? creep
Fail: (13) prince(ann) ? creep
Redo: (12) precedes_new(ann, charles) ? creep
Call: (13) prince(ann) ? creep
Fail: (13) prince(ann) ? creep
Redo: (12) precedes_new(ann, charles) ? creep
Call: (13) princess(ann) ? creep
Exit: (13) princess(ann) ? creep
Call: (13) prince(charles) ? creep
Exit: (13) prince(charles) ? creep
Call: (13) is_older(ann, charles) ? creep
Call: (14) older_than(ann, charles) ? creep
Fail: (14) older_than(ann, charles) ? creep
Redo: (13) is_older(ann, charles) ? creep
Call: (14) older_than(ann, _6436) ? creep
Exit: (14) older_than(ann, andrew) ? creep
Call: (14) older_than(_6434, charles) ? creep
Fail: (14) older_than(_6434, charles) ? creep
Fail: (13) is_older(ann, charles) ? creep
Redo: (12) precedes_new(ann, charles) ? creep
```

```
   Call: (13) princess(ann) ? creep
   Exit: (13) princess(ann) ? creep
   Call: (13) princess(charles) ? creep
   Fail: (13) princess(charles) ? creep
   Fail: (12) precedes_new(ann, charles) ? creep
   Redo: (11) insert_new(charles, [ann, andrew, edward], _6020) ? creep
   Exit: (11) insert_new(charles, [ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
   Exit: (10) succession_order_new([charles, andrew, edward, ann], [charles, ann, andrew, edward]) ? creep
   Exit: (9) succession_list_new(elizabeth, [charles, ann, andrew, edward]) ? creep
Succession_line = [charles, ann, andrew, edward].
```