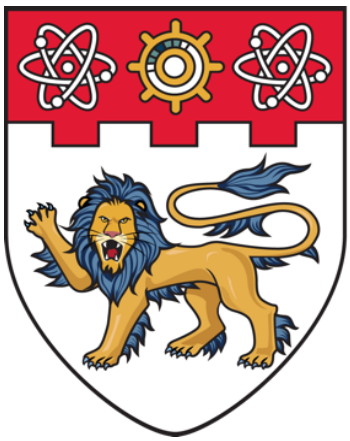

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ3005: Artificial Intelligence

Assignment 1

Shantanu Sharma

U1622895F

Lab Group: SSP3

1. Example 1: FamilyTree.pl

Part 1

Part 2

Knowledge Base System

```
male(jerry).
male(stuart).
male(warren).
male(peter).
female(kather).
female(maryalice).
female(ann).
brother(jerry,stuart).
brother(jerry,kather).
brother(peter, warren).
sister(ann, mayalice).
sister(kather,jerry).
parent_of(warren,jerry).
parent_of(maryalice,jerry).
```

```
brother(peter, warren).
brother(jerry,kather).
brother(jerry,stuart).
male(stuart).
male(peter).
male(warren).
male(jerry).
female(kather).
female(ann).
female(maryalice).
sister(kather,jerry).
sister(ann, mayalice).
parent_of(maryalice,jerry).
parent_of(warren,jerry).
```

```
%Definitions%
father(X,Y):-
    male(X),parent of(X,Y).
mother(X,Y):-
    female(X),parent of(X,Y).
son(X,Y):-
    male(X),parent of(Y,X).
daughter(X,Y):-
    female(X),parent of(Y,X).
grandfather(X,Y):-
    male(X),father(X,Z),parent of(Z,Y).
grandmother(X,Y):-
    female(X),mother(X,Z),parent of(Z,Y).
sibling(X,Y):-
    brother(X,Y);sister(X,Y).
aunt(X,Y):-
    parent of(Z,Y),sister(X,Z).
aunt(X,Y):-
    mother(X,A),cousin(A,Y).
uncle(X,Y):-
    parent of(Z,Y),brother(X,Z).
uncle(X,Y):-
    father(X,A),cousin(A,Y).
cousin(X,Y):-
    parent of(Z,X),sibling(Z,A),parent of(A,Y).
%assuming spouses must have children%
spouse(X,Y):-
   father(X,A),mother(Y,A).
```

Query

```
?- parent_of(X,Y).
X = warren,
Y = jerry
X = maryalice,
Y = jerry.
```

```
?- parent_of(X,Y).
X = maryalice,
Y = jerry
X = warren,
Y = jerry.
```

Tracing

```
[trace]  ?- parent_of(X,Y).
   Call: (8) parent_of(_1610, _1612) ? creep
   Exit: (8) parent_of(warren, jerry) ? creep
X = warren,
Y = jerry
   Redo: (8) parent_of(_1610, _1612) ? creep
   Exit: (8) parent_of(maryalice, jerry) ? creep
X = maryalice,
Y = jerry.
```

```
[trace]  ?- parent_of(X,Y).
   Call: (8) parent_of(_998, _1000) ? creep
   Exit: (8) parent_of(maryalice, jerry) ? creep
X = maryalice,
Y = jerry
   Redo: (8) parent_of(_998, _1000) ? creep
   Exit: (8) parent_of(warren, jerry) ? creep
X = warren,
Y = jerry.
```

Results:

Query- Identical **Trace-** Different

The following result is due to the order of the nodes. Order of nodes matter as they are arranged in the AND-OR Tree and are arranged level by level from left to right before moving on to the next level. Each node(rules) in the derivation tree is a sequence of sub-goals, with edges directly below the node corresponding to choices(facts) available. However, for traversal itself, the operation applies a depth-first search strategy. Thus, it always returns a consistent output of mother, father, mother, father… or father, mother, father, mother…

2. Exercise 1: The Smart Phone Rivalry

The set of First Order Logic for the sentences describing the smart phone industry:

- Competitor (SumSum, Appy)
- Product Technology (GalacticaS3)
- Develop (SumSum, GalacticaS3)
- Boss (Stevey)
- Steal (Stevey, GalacticaS3, SumSum)
- ∀x, y, z Boss (x) ∧ Business (y) ∧ Rival (z) ∧ Steal (x, y, z) → Unethical (x)
- ∀x Competitor (x, Appy) → Rival (x)
- ∀x Product Technology (x) → Business (x)

Equivalent Prolog Statements:

```
competitor(sumsum, appy).
product_technology(galacticaS3).
developed(sumsum, galacticaS3).
steal(stevey, X, sumsum):-product_technology(X), developed(sumsum, X).
boss(stevey).
unethical(X):-boss(X), business(Y), company(Z), rival(Z), steal(X,Y,Z).
rival(X):-competitor(X, appy).
business(X):-product_technology(X).
company(sumsum).
company(appy).
```

To prove that Stevey is unethical, we enter unethical (stevey) into Prolog. The execution trace is as follows:

```
?- edit('smart phone rivalry.pl').
true.

?- reconsult('smart phone rivalry.pl').
true.

?- unethical(X).
X = stevey ;
false.

?- trace.
true.

[trace]  ?- unethical(X).
   Call: (8) unethical(_1804) ? creep
   Call: (9) boss(_1804) ? creep
   Exit: (9) boss(stevey) ? creep
   Call: (9) business(_2018) ? creep
   Call: (10) product_technology(_2018) ? creep
   Exit: (10) product_technology(galacticaS3) ? creep
   Exit: (9) business(galacticaS3) ? creep
   Call: (9) company(_2018) ? creep
   Exit: (9) company(sumsum) ? creep
   Call: (9) rival(sumsum) ? creep
   Call: (10) competitor(sumsum, appy) ? creep
   Exit: (10) competitor(sumsum, appy) ? creep
   Exit: (9) rival(sumsum) ? creep
   Call: (9) steal(stevey, galacticaS3, sumsum) ? creep
   Call: (10) product_technology(galacticaS3) ? creep
   Exit: (10) product_technology(galacticaS3) ? creep
   Call: (10) developed(sumsum, galacticaS3) ? creep
   Exit: (10) developed(sumsum, galacticaS3) ? creep
   Exit: (9) steal(stevey, galacticaS3, sumsum) ? creep
   Exit: (8) unethical(stevey) ? creep
X = stevey ;
   Redo: (9) company(_2018) ? creep
   Exit: (9) company(appy) ? creep
   Call: (9) rival(appy) ? creep
   Call: (10) competitor(appy, appy) ? creep
   Fail: (10) competitor(appy, appy) ? creep
   Fail: (9) rival(appy) ? creep
   Fail: (8) unethical(_1804) ? creep
false.

[trace]  ?-
```

3. Exercise 2: The Royal Family

3.1 Old Succession Rule-

Knowledge Base System:

```
%%old succession rule
precedes(X,Y):- male(X), male(Y), older_than(X,Y).
precedes(X,Y):- male(X), female(Y), Y\=elizabeth.
precedes(X,Y):- female(X), female(Y), older_than(X,Y).


succession_sort([A|B], Sorted) :- succession_sort(B, SortedTail), insert(A, SortedTail, Sorted).
succession_sort([], []).

insert(A, [B|C], [B|D]) :- not(precedes(A,B)), !, insert(A, C, D).
insert(A, C, [A|C]).


successionList(X, SuccessionList):-
        findall(Y, child(Y,X), Children),
        succession_sort(Children, SuccessionList).
```

Tracing:

```
[trace]  ?- successionList(X,Y).
   Call: (8) successionList(_998, _1000) ? creep
^  Call: (9) findall(_1240, child(_1240, _998), _1264) ? creep
   Call: (14) child(_1240, _998) ? creep
   Exit: (14) child(charles, elizabeth) ? creep
   Redo: (14) child(_1240, _998) ? creep
   Exit: (14) child(ann, elizabeth) ? creep
   Redo: (14) child(_1240, _998) ? creep
   Exit: (14) child(andrew, elizabeth) ? creep
   Redo: (14) child(_1240, _998) ? creep
   Exit: (14) child(edward, elizabeth) ? creep
^  Exit: (9) findall(_1240, user:child(_1240, _998), [charles, ann, andrew, edward]) ? creep
   Call: (9) succession_sort([charles, ann, andrew, edward], _1000) ? creep
   Call: (10) succession_sort([ann, andrew, edward], _1320) ? creep
   Call: (11) succession_sort([andrew, edward], _1320) ? creep
   Call: (12) succession_sort([edward], _1320) ? creep
   Call: (13) succession_sort([], _1320) ? creep
   Exit: (13) succession_sort([], []) ? creep
   Call: (13) insert(edward, [], _1322) ? creep
   Exit: (13) insert(edward, [], [edward]) ? creep
   Exit: (12) succession_sort([edward], [edward]) ? creep
   Call: (12) insert(andrew, [edward], _1328) ? creep
^  Call: (13) not(precedes(andrew, edward)) ? creep
   Call: (14) precedes(andrew, edward) ? creep
   Call: (15) male(andrew) ? creep
   Exit: (15) male(andrew) ? creep
   Call: (15) male(edward) ? creep
   Exit: (15) male(edward) ? creep
   Call: (15) older_than(andrew, edward) ? creep
   Exit: (15) older_than(andrew, edward) ? creep
   Exit: (14) precedes(andrew, edward) ? creep
^  Fail: (13) not(user:precedes(andrew, edward)) ? creep
   Redo: (12) insert(andrew, [edward], _1328) ? creep
   Exit: (12) insert(andrew, [edward], [andrew, edward]) ? creep
   Exit: (11) succession_sort([andrew, edward], [andrew, edward]) ? creep
   Call: (11) insert(ann, [andrew, edward], _1334) ? creep
^  Call: (12) not(precedes(ann, andrew)) ? creep
   Call: (13) precedes(ann, andrew) ? creep
   Call: (14) male(ann) ? creep
   Fail: (14) male(ann) ? creep
   Redo: (13) precedes(ann, andrew) ? creep
   Call: (14) male(ann) ? creep
   Fail: (14) male(ann) ? creep
   Redo: (13) precedes(ann, andrew) ? creep
   Call: (14) female(ann) ? creep
   Exit: (14) female(ann) ? creep
   Call: (14) female(andrew) ? creep
   Fail: (14) female(andrew) ? creep
   Fail: (13) precedes(ann, andrew) ? creep
^  Exit: (12) not(user:precedes(ann, andrew)) ? creep
   Call: (12) insert(ann, [edward], _1318) ? creep
^  Call: (13) not(precedes(ann, edward)) ? creep
   Call: (14) precedes(ann, edward) ? creep
   Call: (15) male(ann) ? creep
   Fail: (15) male(ann) ? creep
   Redo: (14) precedes(ann, edward) ? creep
   Call: (15) male(ann) ? creep
   Fail: (15) male(ann) ? creep
   Redo: (14) precedes(ann, edward) ? creep
   Call: (15) female(ann) ? creep
   Exit: (15) female(ann) ? creep
   Call: (15) female(edward) ? creep
   Fail: (15) female(edward) ? creep
   Fail: (14) precedes(ann, edward) ? creep
^  Exit: (13) not(user:precedes(ann, edward)) ? creep
   Call: (13) insert(ann, [], _1336) ? creep
   Exit: (13) insert(ann, [], [ann]) ? creep
   Exit: (12) insert(ann, [edward], [edward, ann]) ? creep
   Exit: (11) insert(ann, [andrew, edward], [andrew, edward, ann]) ? creep
   Exit: (10) succession_sort([ann, andrew, edward], [andrew, edward, ann]) ? creep
   Call: (10) insert(charles, [andrew, edward, ann], _1000) ? creep
^  Call: (11) not(precedes(charles, andrew)) ? creep
   Call: (12) precedes(charles, andrew) ? creep
   Call: (13) male(charles) ? creep
   Exit: (13) male(charles) ? creep
   Call: (13) male(andrew) ? creep
   Exit: (13) male(andrew) ? creep
   Call: (13) older_than(charles, andrew) ? creep
   Exit: (13) older_than(charles, andrew) ? creep
   Exit: (12) precedes(charles, andrew) ? creep
^  Fail: (11) not(user:precedes(charles, andrew)) ? creep
   Redo: (10) insert(charles, [andrew, edward, ann], _1000) ? creep
   Exit: (10) insert(charles, [andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (9) succession_sort([charles, ann, andrew, edward], [charles, andrew, edward, ann]) ? creep
   Exit: (8) successionList(_998, [charles, andrew, edward, ann]) ? creep
Y = [charles, andrew, edward, ann].
```

3.2 New Succession Rule-

Knowledge Base System:

```
%%new succession rule - irregardless of gender
successor(X, Y):- child(Y,X).

successionListIndependent(X, SuccessionList):-
        findall(Y, successor(X, Y), SuccessionList).
```

Tracing:

```
[trace]  ?- successionListIndependent(X,Y).
   Call: (8) successionListIndependent(_998, _1000) ? creep
^  Call: (9) findall(_1248, successor(_998, _1248), _1000) ? creep
   Call: (14) successor(_998, _1248) ? creep
   Call: (15) child(_1248, _998) ? creep
   Exit: (15) child(charles, elizabeth) ? creep
   Exit: (14) successor(elizabeth, charles) ? creep
   Redo: (15) child(_1248, _998) ? creep
   Exit: (15) child(ann, elizabeth) ? creep
   Exit: (14) successor(elizabeth, ann) ? creep
   Redo: (15) child(_1248, _998) ? creep
   Exit: (15) child(andrew, elizabeth) ? creep
   Exit: (14) successor(elizabeth, andrew) ? creep
   Redo: (15) child(_1248, _998) ? creep
   Exit: (15) child(edward, elizabeth) ? creep
   Exit: (14) successor(elizabeth, edward) ? creep
^  Exit: (9) findall(_1248, user:successor(_998, _1248), [charles, ann, andrew, edward]) ? creep
   Exit: (8) successionListIndependent(_998, [charles, ann, andrew, edward]) ? creep
Y = [charles, ann, andrew, edward].
```

```
[trace]  ?- successionList(X,Y).
   Call: (8) successionList(_998, _1000) ? creep
^  Call: (9) findall(_1240, child(_1240, _998), _1264) ? creep
   Call: (14) child(_1240, _998) ? creep
   Exit: (14) child(charles, elizabeth) ? creep
   Redo: (14) child(_1240, _998) ? creep
   Exit: (14) child(ann, elizabeth) ? creep
   Redo: (14) child(_1240, _998) ? creep
   Exit: (14) child(andrew, elizabeth) ? creep
   Redo: (14) child(_1240, _998) ? creep
   Exit: (14) child(edward, elizabeth) ? creep
^  Exit: (9) findall(_1240, user:child(_1240, _998), [charles, ann, andrew, edward]) ? creep
   Call: (9) succession_sort([charles, ann, andrew, edward], _1000) ? creep
   Call: (10) succession_sort([ann, andrew, edward], _1320) ? creep
   Call: (11) succession_sort([andrew, edward], _1320) ? creep
   Call: (12) succession_sort([edward], _1320) ? creep
   Call: (13) succession_sort([], _1320) ? creep
   Exit: (13) succession_sort([], []) ? creep
   Call: (13) insert(edward, [], _1322) ? creep
   Exit: (13) insert(edward, [], [edward]) ? creep
   Exit: (12) succession_sort([edward], [edward]) ? creep
   Call: (12) insert(andrew, [edward], _1328) ? creep
^  Call: (13) not(precedes(andrew, edward)) ? creep
   Call: (14) precedes(andrew, edward) ? creep
   Call: (15) male(andrew) ? creep
   Exit: (15) male(andrew) ? creep
   Call: (15) male(edward) ? creep
   Exit: (15) male(edward) ? creep
   Call: (15) older_than(andrew, edward) ? creep
   Exit: (15) older_than(andrew, edward) ? creep
   Exit: (14) precedes(andrew, edward) ? creep
^  Fail: (13) not(user:precedes(andrew, edward)) ? creep
   Fail: (13) precedes(ann, andrew) ? creep
^  Exit: (12) not(user:precedes(ann, andrew)) ? creep
   Call: (12) insert(ann, [edward], _1318) ? creep
^  Call: (13) not(precedes(ann, edward)) ? creep
   Call: (14) precedes(ann, edward) ? creep
   Call: (15) male(ann) ? creep
   Fail: (15) male(ann) ? creep
   Redo: (14) precedes(ann, edward) ? creep
   Call: (15) male(ann) ? creep
   Fail: (15) male(ann) ? creep
   Redo: (14) precedes(ann, edward) ? creep
   Call: (15) female(ann) ? creep
   Exit: (15) female(ann) ? creep
   Call: (15) female(edward) ? creep
   Fail: (15) female(edward) ? creep
   Fail: (14) precedes(ann, edward) ? creep
^  Exit: (13) not(user:precedes(ann, edward)) ? creep
   Call: (13) insert(ann, [], _1336) ? creep
   Exit: (13) insert(ann, [], [ann]) ? creep
   Exit: (12) insert(ann, [edward], [edward, ann]) ? creep
   Exit: (11) insert(ann, [andrew, edward], [andrew, edward, ann]) ? creep
   Exit: (10) succession_sort([ann, andrew, edward], [andrew, edward, ann]) ? creep
   Call: (10) insert(charles, [andrew, edward, ann], _1000) ? creep
^  Call: (11) not(precedes(charles, andrew)) ? creep
   Call: (12) precedes(charles, andrew) ? creep
   Call: (13) male(charles) ? creep
   Exit: (13) male(charles) ? creep
   Call: (13) male(andrew) ? creep
   Exit: (13) male(andrew) ? creep
   Call: (13) older_than(charles, andrew) ? creep
   Exit: (13) older_than(charles, andrew) ? creep
   Exit: (12) precedes(charles, andrew) ? creep
^  Fail: (11) not(user:precedes(charles, andrew)) ? creep
   Redo: (10) insert(charles, [andrew, edward, ann], _1000) ? creep
   Exit: (10) insert(charles, [andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
   Exit: (9) succession_sort([charles, ann, andrew, edward], [charles, andrew, edward, ann]) ? creep
   Exit: (8) successionList(_998, [charles, andrew, edward, ann]) ? creep
Y = [charles, andrew, edward, ann].
```

Explanation:

In the old succession rule, since we had to factor in not only the order of birth, but also the gender, thus for every new variable/element, we must perform a comparison with the elements in the ArrayList that we initialize to temporarily hold the succession order list as it builds up and determine where to do the insertion. This operation can only be performed one at a time.

Changing the Royal Succession rule, there is no longer a need for us to rearrange the order of the succession base on gender. Hence, we can do away the precedence of gender over order of birth in the old succession rule. Thus, the new royal succession rule will simply just consider the order of birth, which only factors in age as consideration.

However, for the new succession order, there isn't a need to implement an ArrayList as we ca simply determine the order of succession base