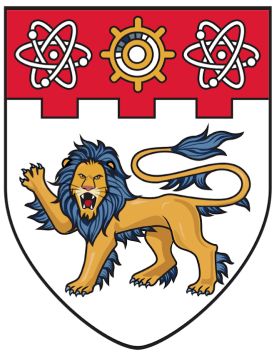


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC3000 Artificial Intelligence

Lab Assignment 2:
Introduction to Prolog

1.	S Ramaneesen	U2323603K
2.	Mishra Aviral	U2323862A

TABLE OF CONTENTS

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.

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

Natural Language	**First Order Logic (FOL)**
sumsum, a competitor of appy	competitor(sumsum, appy)
sumsum developed some nice smart phone technology called galactica-s3	developed(sumsum, galactica-s3)
smart phone technology called galactica-s3	smartphonetech(galactica-s3)
stevey stole galactica-s3	stole(stevey, galactica-s3)
stevey is a boss of appy	boss(stevey, appy)

Rules	First Order Logic (FOL)
A competitor is a rival	$\forall x \ \forall y$ (competitor(x, y) $\rightarrow$ rival(x, y))
unethical for a boss to steal business from rival companies	$\forall x \ \forall y \ \forall z \ \forall b$ ((boss(x, y) $\wedge$ rival(z, y) $\wedge$ business(b) $\wedge$ stole(x, b) $\wedge$ developed(z, b)) $\rightarrow$ unethical(x))
Smart phone technology is business	$\forall x$ (smartphonetech(x) $\rightarrow$ business(x))

1.2 Write these FOL statements as Prolog clauses.

```
% Facts
competitor(sumsum, appy).
developed(sumsum, galacticas3).
smartphonetech(galacticas3).
stole(stevey, galacticas3).
boss(stevey, appy).

% Rules
rival(X, Y) :- competitor(X, Y).
business(X) :- smartphonetech(X).
unethical(X) :- boss(X, Y), rival(Z, Y), business(B), stole(X, B), developed(Z, B).
```

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

```
?- trace, unethical(stevey).
   Call: (13) unethical(stevey) ? creep
   Call: (14) boss(stevey, _26286) ? creep
   Exit: (14) boss(stevey, appy) ? creep
   Call: (14) rival(_27908, appy) ? creep
   Call: (15) competitor(_27908, appy) ? creep
   Exit: (15) competitor(sumsum, appy) ? creep
   Exit: (14) rival(sumsum, appy) ? creep
   Call: (14) business(_31150) ? creep
   Call: (15) smartphonetech(_31150) ? creep
   Exit: (15) smartphonetech(galacticas3) ? creep
   Exit: (14) business(galacticas3) ? creep
   Call: (14) stole(stevey, galacticas3) ? creep
   Exit: (14) stole(stevey, galacticas3) ? creep
   Call: (14) developed(sumsum, galacticas3) ? creep
   Exit: (14) developed(sumsum, galacticas3) ? 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 off-springs; namely:- prince charles, princess ann, prince andrew and prince edward – listed in the order of birth.

2.1 Define their relations and succession using old rules in a Prolog rule base.

In the old royal succession rule, males are given preference over females. Our Prolog program models this by:

- Defining male/1 and female/1 for gender.
- Using born/2 to capture birth order (lower number = earlier birth).
- Creating rules male_successor/1 and female_successor/1 to identify eligible children of Queen Elizabeth.
- Sorting the males and females separately using keysort/2 based on birth order.
- Finally, using append/3 to combine the male list followed by the female list, giving us the full line of succession.

Full Prolog code (Q2.1):

royal_old.pl

Users > aviral > Downloads > royal_old.pl

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

% Birth order (lower number = earlier)
born(prince_charles, 1).
born(princess_ann, 2).
born(prince_andrew, 3).
born(prince_edward, 4).

% Children of Queen Elizabeth
child(prince_charles, queen_elizabeth).
child(princess_ann, queen_elizabeth).
child(prince_andrew, queen_elizabeth).
child(prince_edward, queen_elizabeth).

% Males succeed first
male_successor(X) :-
    male(X),
    child(X, queen_elizabeth).

% Females succeed after all males
female_successor(X) :-
    female(X),
    child(X, queen_elizabeth).

% Collect full succession list: males first (ordered), then females (ordered)
ordered_succession(List) :-
    findall(B-Male, (male_successor(Male), born(Male, B)), Males),
    keysort(Males, SortedMales),
    pairs_values(SortedMales, MaleList),

    findall(B-Female, (female_successor(Female), born(Female, B)), Females),
    keysort(Females, SortedFemales),
    pairs_values(SortedFemales, FemaleList),

    append(MaleList, FemaleList, List).
```

The screenshot below shows the result of running:
?- ordered_succession(List).

```
% /Users/aviral/Downloads/royal_old.pl compiled 0.00 sec, 15 clauses
?- ordered_succession(List).

List = [prince_charles, prince_andrew, prince_edward, princess_ann].
```

We also performed a trace using:
trace.
ordered_succession(List).

```
[trace]  ?- ordered_succession(List).
   Call: (12) ordered_succession(_31540) ? creep
^  Call: (13) findall(_33192-_33194, (male_successor(_33194), born(_33194, _33192)), _33212) ? creep
   Call: (18) male_successor(_33194) ? creep
   Call: (19) male(_33194) ? creep
   Exit: (19) male(prince_charles) ? creep
   Call: (19) child(prince_charles, queen_elizabeth) ? creep
   Exit: (19) child(prince_charles, queen_elizabeth) ? creep
   Exit: (18) male_successor(prince_charles) ? creep
   Call: (18) born(prince_charles, _33192) ? creep
   Exit: (18) born(prince_charles, 1) ? creep
   Redo: (19) male(_33194) ? creep
   Exit: (19) male(prince_andrew) ? creep
   Call: (19) child(prince_andrew, queen_elizabeth) ? creep
   Exit: (19) child(prince_andrew, queen_elizabeth) ? creep
   Exit: (18) male_successor(prince_andrew) ? creep
   Call: (18) born(prince_andrew, _33192) ? creep
   Exit: (18) born(prince_andrew, 3) ? creep
   Redo: (19) male(_33194) ? creep
   Exit: (19) male(prince_edward) ? creep
   Call: (19) child(prince_edward, queen_elizabeth) ? creep
   Exit: (19) child(prince_edward, queen_elizabeth) ? creep
   Exit: (18) male_successor(prince_edward) ? creep
   Call: (18) born(prince_edward, _33192) ? creep
   Exit: (18) born(prince_edward, 4) ? creep
^  Exit: (13) findall(_33192-_33194, user:(male_successor(_33194), born(_33194, _33192)), [1-prince_charles, 3-prince_andrew, 4-prince_edward]) ? creep
   Call: (13) keysort([1-prince_charles, 3-prince_andrew, 4-prince_edward], _54930) ? creep
   Exit: (13) keysort([1-prince_charles, 3-prince_andrew, 4-prince_edward], [1-prince_charles, 3-prince_andrew, 4-prince_edward]) ? creep
   Call: (13) pairs:pairs_values([1-prince_charles, 3-prince_andrew, 4-prince_edward], _56754) ? creep
   Call: (14) pairs:pairs_values([3-prince_andrew, 4-prince_edward], _57674) ? creep
   Call: (15) pairs:pairs_values([4-prince_edward], _58594) ? creep
   Call: (16) pairs:pairs_values([], _59514) ? creep
   Exit: (16) pairs:pairs_values([], []) ? creep
   Exit: (15) pairs:pairs_values([4-prince_edward], [prince_edward]) ? creep
   Exit: (14) pairs:pairs_values([3-prince_andrew, 4-prince_edward], [prince_andrew, prince_edward]) ? creep
   Exit: (13) pairs:pairs_values([1-prince_charles, 3-prince_andrew, 4-prince_edward], [prince_charles, prince_andrew, prince_edward]) ? creep
^  Call: (13) findall(_33192-_64090, (female_successor(_64090), born(_64090, _33192)), _64108) ? creep
   Call: (18) female_successor(_64090) ? creep
   Call: (19) female(_64090) ? creep
   Exit: (19) female(princess_ann) ? creep
   Call: (19) child(princess_ann, queen_elizabeth) ? creep
   Exit: (19) child(princess_ann, queen_elizabeth) ? creep
   Exit: (18) female_successor(princess_ann) ? creep
   Call: (18) born(princess_ann, _116) ? creep
   Exit: (18) born(princess_ann, 2) ? creep
^  Exit: (13) findall(_116-_202, user:(female_successor(_202), born(_202, _116)), [2-princess_ann]) ? creep
   Call: (13) keysort([2-princess_ann], _5168) ? creep
   Exit: (13) keysort([2-princess_ann], [2-princess_ann]) ? creep
   Call: (13) pairs:pairs_values([2-princess_ann], _6980) ? creep
   Call: (14) pairs:pairs_values([], _7900) ? creep
   Exit: (14) pairs:pairs_values([], []) ? creep
   Exit: (13) pairs:pairs_values([2-princess_ann], [princess_ann]) ? creep
   Call: (13) lists:append([prince_charles, prince_andrew, prince_edward], [princess_ann], _60) ? creep
   Exit: (13) lists:append([prince_charles, prince_andrew, prince_edward], [princess_ann], [prince_charles, prince_andrew, prince_edward, princess_ann]) ? creep
   Exit: (12) ordered_succession([prince_charles, prince_andrew, prince_edward, princess_ann]) ? creep
List = [prince_charles, prince_andrew, prince_edward, princess_ann].
```

The output confirms that males are prioritized in birth order, followed by the females.

2.2 Define their relations and succession using new rules in a Prolog rule base.

In the updated royal succession rule, **gender is no longer a factor**. Succession is determined **strictly by birth order**, regardless of whether the heir is male or female.

Our Prolog program models this rule as follows:

1. **Define Birth Order**

Each child of Queen Elizabeth is assigned a birth order using the born/2 predicate. A lower number indicates earlier birth.

```
% Birth order: smaller number = born earlier
born(prince_charles, 1).
born(princess_ann, 2).
born(prince_andrew, 3).
born(prince_edward, 4).
```

2. **Define Parent-Child Relationships**

The child/2 predicate specifies each offspring of Queen Elizabeth.

```
% Children of Queen Elizabeth
child(prince_charles, queen_elizabeth).
child(princess_ann, queen_elizabeth).
child(prince_andrew, queen_elizabeth).
child(prince_edward, queen_elizabeth).
```

3. **Generalize the Successor Rule**

The successor/1 rule includes any child of Queen Elizabeth who has a known birth order, regardless of gender.

```
% In the new succession rule, any child of the queen is a valid successor,
% regardless of gender. We also check that the birth order is known.
successor(X) :-
    child(X, queen_elizabeth),
    born(X, _).
```

4. **Generate Succession List by Birth Order**

The ordered_new_succession/1 rule:

- Collects child–birth order pairs using findall/3

- Sorts them using keysort/2
- Extracts the names in the correct order using pairs_values/2

```
% This rule generates the succession list based on birth order only.
% It collects child-birth pairs, sorts them by birth number, and extracts the names.
ordered_new_succession(List) :-
    findall(Order-Name, (successor(Name), born(Name, Order)), Pairs),
    keysort(Pairs, SortedPairs),
    pairs_values(SortedPairs, List).
```

Full Prolog Code (Q2.2):

royal_old.pl | royal_new.pl

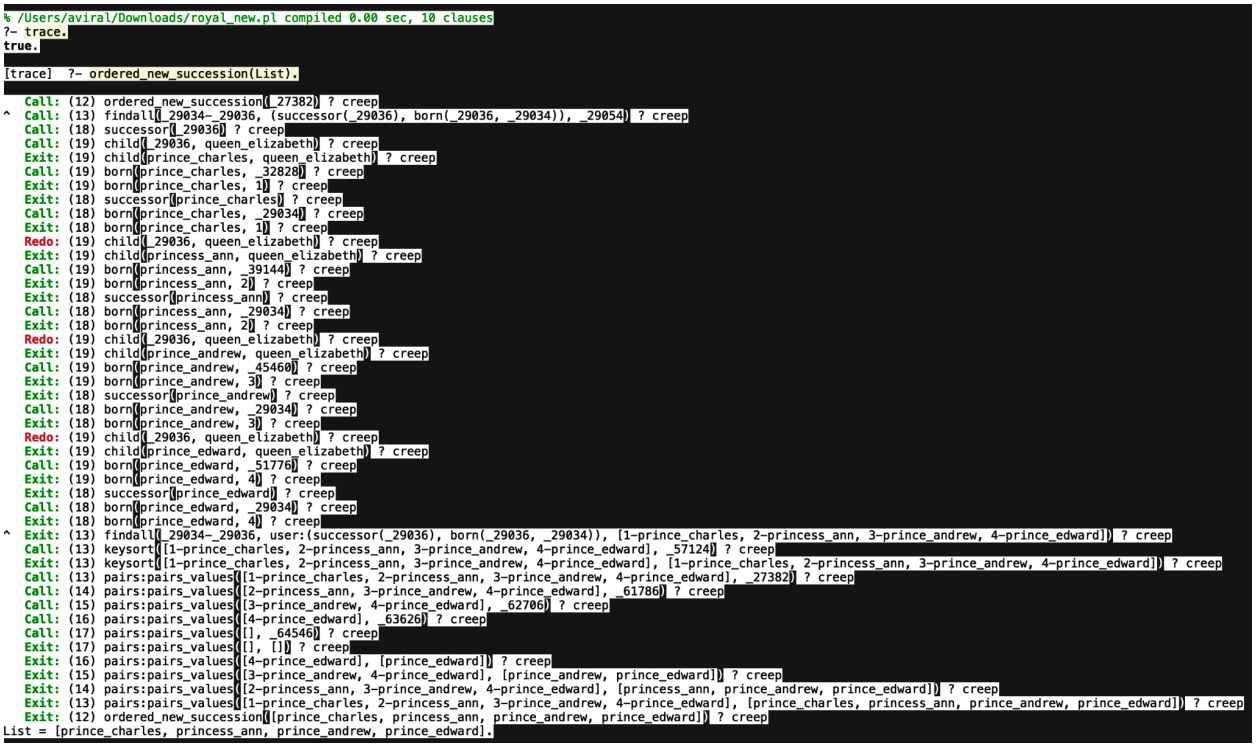
Users > aviral > Downloads > royal_new.pl

```
% Birth order: smaller number = born earlier
born(prince_charles, 1).
born(princess_ann, 2).
born(prince_andrew, 3).
born(prince_edward, 4).

% Children of Queen Elizabeth
child(prince_charles, queen_elizabeth).
child(princess_ann, queen_elizabeth).
child(prince_andrew, queen_elizabeth).
child(prince_edward, queen_elizabeth).

% In the new succession rule, any child of the queen is a valid successor,
% regardless of gender. We also check that the birth order is known.
successor(X) :-
    child(X, queen_elizabeth),
    born(X, _).

% This rule generates the succession list based on birth order only.
% It collects child-birth pairs, sorts them by birth number, and extracts the names.
ordered_new_succession(List) :-
    findall(Order-Name, (successor(Name), born(Name, Order)), Pairs),
    keysort(Pairs, SortedPairs),
    pairs_values(SortedPairs, List).
```

The screenshot (see below) shows the result of running:
?- ordered_new_succession(List).

It returns:

```
List = [prince_charles, princess_ann, prince_andrew, prince_edward].
```



The output confirms that all children are ordered purely by birth, regardless of gender, reflecting the new succession rule.

Conclusion

This assignment deepened our understanding of logic programming using Prolog. We modeled real-world scenarios through declarative rules and explored how inference works in knowledge-based systems.

In Exercise 1, we applied logical reasoning to identify unethical behavior. In Exercise 2, we modeled royal succession under both traditional and modern rules, using Prolog to generate and trace ordered successors.

Overall, this lab reinforced key AI concepts like rule-based reasoning, logical inference, and the impact of rule structure on outcomes.