SC3000: Artificial Intelligence
Assignment 2 Report: TDDB Team Armaan & Friends

Members:

Goel Armaan (U2123642H)
Exercise 1.1, 1.2, 1.3, 2-Alt

Chiam Da Jie (U2121233G)
Exercise 2.1

Guo Yuan Lin (U2122626F)
Exercise 2.2

We assert that everyone has contributed equally to the creation of this report.

Date of Submission: November 12, 2023

References

[1] "Prolog – finding all solutions to a goal", *SWI Prolog*, 02-Oct-2013. [Online]. Available: https://www.swi-prolog.org/pldoc/man?section=allsolutions. [Accessed: 12-Nov-2023]

1 The Smart Phone Rivalry

1.1 Translating to First Order Logic (FOL)

1. “sumsum, a competitor of appy”
$Company(sumsum)$
$Company(appy)$
$Competitor(sumsum, appy)$

2. “developed some nice smart phone technology called galactica-s3”
$SmartPhoneTech(galactica-s3)$
$Developed(sumsum, galactica-s3)$

3. “all of which was stolen by stevey”
$Stole(stevey, galactica-s3)$

4. “who is a boss of appy”
$Boss(stevey, appy)$

5. “A competitor is a rival”
$Competitor(X, Y) \implies Competitor(Y, X)$
$Competitor(X, Y) \implies Rival(X, Y)$

6. “Smart phone technology is business”
$SmartPhoneTech(X) \implies Business(X)$

7. “It is unethical for a boss to steal business from rival companies”
$Boss(B, C) \land Company(C) \land Stole(B, T) \land Business(T) \land Developed(R, T) \land Rival(C, R) \land Company(R) \implies Unethical(B)$

1.2 Converting to Prolog Clauses

Please see *Armaan and Friends_qn_1_2.pl*

1.3 Prolog Trace

```
[trace] ?- unethical(stevey).
   Call: (10) unethical(stevey) ? creep
   Call: (11) boss(stevey, _10468) ? creep
   Exit: (11) boss(stevey, appy) ? creep
   Call: (11) company(appy) ? creep
   Exit: (11) company(appy) ? creep
   Call: (11) stole(stevey, _13702) ? creep
   Exit: (11) stole(stevey, galactica-s3) ? creep
   Call: (11) business(galactica-s3) ? creep
```

```
  Call: (12) smartphonetech(galactica-s3) ? creep
  Exit: (12) smartphonetech(galactica-s3) ? creep
  Exit: (11) business(galactica-s3) ? creep
  Call: (11) developed(_18554, galactica-s3) ? creep
  Exit: (11) developed(sumsum, galactica-s3) ? creep
  Call: (11) rival(appy, sumsum) ? creep
  Call: (12) competitor(appy, sumsum) ? creep
  Exit: (12) competitor(appy, sumsum) ? creep
  Exit: (11) rival(appy, sumsum) ? creep
  Call: (11) company(sumsum) ? creep
  Exit: (11) company(sumsum) ? creep
  Exit: (10) unethical(stevey) ? creep
true .
```

2 The Royal Family

2.1 Old Succession Rule

Please see *Armaan and Friends_qn_2_1.pl* for the prolog rules and relations.

Prolog Trace:

```
[trace] ?- succession(elizabeth).
  Call: (10) succession(elizabeth) ? creep
  Call: (11) monarch(elizabeth) ? creep
  Exit: (11) monarch(elizabeth) ? creep
  Call: (11) setof(_6836-_6838, son(elizabeth, _6838, _6836), _6848) ? creep
  Call: (17) son(elizabeth, _6838, _6836) ? creep
  Exit: (17) son(elizabeth, charles, 1) ? creep
  Redo: (17) son(elizabeth, _6838, _6836) ? creep
  Exit: (17) son(elizabeth, andrew, 3) ? creep
  Redo: (17) son(elizabeth, _6838, _6836) ? creep
  Exit: (17) son(elizabeth, edward, 4) ? creep
  Exit: (11) setof(_6836-_6838, user:son(elizabeth, _6838, _6836), [1-charles, 3-andrew, 4-edward]) ? creep
  Call: (11) setof(_6836-_6838, daughter(elizabeth, _6838, _6836), _13506) ? creep
  Call: (17) daughter(elizabeth, _6838, _6836) ? creep
  Exit: (17) daughter(elizabeth, ann, 2) ? creep
  Exit: (11) setof(_6836-_6838, user:daughter(elizabeth, _6838, _6836), [2-ann]) ? creep
  Call: (11) print_list([1-charles, 3-andrew, 4-edward]) ? creep
  Call: (12) writeln(charles) ? creep
charles
  Exit: (12) writeln(charles) ? creep
  Call: (12) print_list([3-andrew, 4-edward]) ? creep
  Call: (13) writeln(andrew) ? creep
```

```
andrew
  Exit: (13) writeln(andrew) ? creep
  Call: (13) print_list([4-edward]) ? creep
  Call: (14) writeln(edward) ? creep
edward
  Exit: (14) writeln(edward) ? creep
  Call: (14) print_list([]) ? creep
  Exit: (14) print_list([]) ? creep
  Exit: (13) print_list([4-edward]) ? creep
  Exit: (12) print_list([3-andrew, 4-edward]) ? creep
  Exit: (11) print_list([1-charles, 3-andrew, 4-edward]) ? creep
  Call: (11) print_list([2-ann]) ? creep
  Call: (12) writeln(ann) ? creep
ann
  Exit: (12) writeln(ann) ? creep
  Call: (12) print_list([]) ? creep
  Exit: (12) print_list([]) ? creep
  Exit: (11) print_list([2-ann]) ? creep
  Exit: (10) succession(elizabeth) ? creep
true.
```

2.2 New Succession Rule

Please see *Armaan and Friends_qn_2_2.pl* for the prolog rules and relations.

Changes to knowledge needed:
The facts don't need to be changed for the new succession rule. The only difference is that we find a set of all the children regardless of gender at once.

Prolog Trace:

```
[trace] ?- succession(elizabeth).
  Call: (10) succession(elizabeth) ? creep
  Call: (11) monarch(elizabeth) ? creep
  Exit: (11) monarch(elizabeth) ? creep
  Call: (11) setof(_6836-_6838, (son(elizabeth, _6838, _6836);daughter(elizabeth, _6838, _6836)), _6862) ? creep
  Call: (18) son(elizabeth, _6838, _6836) ? creep
  Exit: (18) son(elizabeth, charles, 1) ? creep
  Redo: (18) son(elizabeth, _6838, _6836) ? creep
  Exit: (18) son(elizabeth, andrew, 3) ? creep
  Redo: (18) son(elizabeth, _6838, _6836) ? creep
  Exit: (18) son(elizabeth, edward, 4) ? creep
  Call: (18) daughter(elizabeth, _6838, _6836) ? creep
  Exit: (18) daughter(elizabeth, ann, 2) ? creep
  Exit: (11) setof(_6836-_6838, user:(son(elizabeth, _6838, _6836);daughter(elizabeth,
_6838, _6836)), [1-charles, 2-ann, 3-andrew, 4-edward]) ? creep
   Call: (11) print_list([1-charles, 2-ann, 3-andrew, 4-edward]) ? creep
   Call: (12) writeln(charles) ? creep
charles
   Exit: (12) writeln(charles) ? creep
   Call: (12) print_list([2-ann, 3-andrew, 4-edward]) ? creep
   Call: (13) writeln(ann) ? creep
ann
   Exit: (13) writeln(ann) ? creep
   Call: (13) print_list([3-andrew, 4-edward]) ? creep
   Call: (14) writeln(andrew) ? creep
andrew
   Exit: (14) writeln(andrew) ? creep
   Call: (14) print_list([4-edward]) ? creep
   Call: (15) writeln(edward) ? creep
edward
   Exit: (15) writeln(edward) ? creep
   Call: (15) print_list([]) ? creep
   Exit: (15) print_list([]) ? creep
   Exit: (14) print_list([4-edward]) ? creep
   Exit: (13) print_list([3-andrew, 4-edward]) ? creep
   Exit: (12) print_list([2-ann, 3-andrew, 4-edward]) ? creep
   Exit: (11) print_list([1-charles, 2-ann, 3-andrew, 4-edward]) ? creep
   Exit: (10) succession(elizabeth) ? creep
true.
```

2.3 Alternative

There is a simpler alternative that can be used to implement both succession rules.
(See *Armaan and Friends_qn_2_alt.pl*)

However, it will only work if the children are added to our knowledge base in the order of their birth. Moreover, the entire knowledge base will also have to be rebuilt to accommodate the change in succession rule. For these reasons we have not chosen this implementation as our main answer, but we are still including it to showcase how different approaches can be taken to solve the same problem and to highlight the importance of efficient knowledge representation and designing robust rules.

Prolog Trace (for reference):

```
[trace] ?- succession_old(elizabeth,C).
   Call: (10) succession_old(elizabeth, _14118) ? creep
   Call: (11) son(elizabeth, _14118) ? creep
   Exit: (11) son(elizabeth, charles) ? creep
   Exit: (10) succession_old(elizabeth, charles) ? creep
C = charles ;
```

```
  Redo: (11) son(elizabeth, _14118) ? creep
  Exit: (11) son(elizabeth, andrew) ? creep
  Exit: (10) succession_old(elizabeth, andrew) ? creep
C = andrew ;
  Redo: (11) son(elizabeth, _14118) ? creep
  Exit: (11) son(elizabeth, edward) ? creep
  Exit: (10) succession_old(elizabeth, edward) ? creep
C = edward ;
  Redo: (10) succession_old(elizabeth, _14118) ? creep
  Call: (11) daughter(elizabeth, _14118) ? creep
  Exit: (11) daughter(elizabeth, ann) ? creep
  Exit: (10) succession_old(elizabeth, ann) ? creep
C = ann.

[trace] ?- succession_new(elizabeth,C).
  Call: (10) succession_new(elizabeth, _40882) ? creep
  Call: (11) offspring(elizabeth, _40882) ? creep
  Exit: (11) offspring(elizabeth, charles) ? creep
  Exit: (10) succession_new(elizabeth, charles) ? creep
C = charles ;
  Redo: (11) offspring(elizabeth, _40882) ? creep
  Exit: (11) offspring(elizabeth, ann) ? creep
  Exit: (10) succession_new(elizabeth, ann) ? creep
C = ann ;
  Redo: (11) offspring(elizabeth, _40882) ? creep
  Exit: (11) offspring(elizabeth, andrew) ? creep
  Exit: (10) succession_new(elizabeth, andrew) ? creep
C = andrew ;
  Redo: (11) offspring(elizabeth, _40882) ? creep
  Exit: (11) offspring(elizabeth, edward) ? creep
  Exit: (10) succession_new(elizabeth, edward) ? creep
C = edward.
```