NANYANG TECHNOLOGICAL UNIVERSITY

CZ3005 Artificial Intelligence

Assignment 3: Introduction to Prolog

Date of Submission:
27/10/2020

Lab Group: TSP2

Submitted By:
Royce Ang Jia Jie
U1840416D

Exercise 1

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

Constants:

1. Sumsum
2. Appy
3. Galacticas3
4. Stevey

Predicate	**Definition**
Competitor(x,y)	X is a competitor of Y.
Boss(x)	X is a boss.
Rival(x)	X is a rival.
Unethical(x)	X is unethical.
Business(x)	X is a business.
Developed(x,y)	X developed Y.
Steal(x,y,z)	X steals Y from Z. Where Y is a smartphone developed by Z.
Company(x)	X is a company.
SmartPhoneTech(x)	X is a smart phone technology.

S/N	**Natural Language**	**First Order Logic**
1	"Sumsum, a competitor of Appy"	Competitor(Sumsum, Appy) Company(Sumsum) Company (Appy)
2	"Sumsum developed some nice smart phone technology called Galaticas3."	Developed(Sumsum, Galaticas3) SmartPhoneTech(Galaticas3)
3	"All of which(Galaticas3) was stolen by Stevey"	∀x SmartPhoneTech(x) ^ Developed(Sumsum, x) ⇒ Steal(Stevey, x, Sumsum)
4	"Stevey, who is a boss"	Boss(Stevey)
5	"It is unethical for a boss to steal business from rival companies"	∀x, y, z Boss(x) ^ Business(y) ^ Rival(z) ^ Steal(x,y,z) ⇒ Unethical(x)
6	"A competitor of Appy is a rival"	∀x Competitor(x, Appy) ⇒ Rival(x)
7	"Smart phone technology is a business"	∀x SmartPhoneTech(x) ⇒ Business(x)

Assumptions:

1. Competitor(x,y) ⇒ Competitor(y,x): if x is a competitor of y, then y is also a competitor of x.

Exercise 1

1.2 Write these FOL statements as Prolog clauses.

S/N	First Order Logic	Prolog Clauses	Rules/Fact
1	Competitor(Sumsum, Appy) Company(Sumsum) Company (Appy)	competitor(sumsum,appy). company(sumsum). company(appy).	Fact
2	Developed(Sumsum, Galaticas3) SmartPhoneTech(Galaticas3)	developed(sumsum, galatica-s3). smartPhoneTech(galatica-s3).	Fact
3	∀x SmartPhoneTech(x) ^ Developed(Sumsum, x) ⇒ Steal(Stevey, x, Sumsum)	steal(stevey, X, sumsum) :- smartphonetech(X), developed(sumsum,X).	Rule
4	Boss(Stevey)	boss(stevey).	Fact
5	∀x, y, z Boss(x) ^ Business(y) ^ Rival(z) ^ Steal(x,y,z) ⇒ Unethical(x)	unethical(X) :- boss(X), business(Y), rival(Z), steal(X,Y,Z).	Rule
6	∀x Competitor(x, Appy) ⇒ Rival(x)	rival(X) :- competitor(X,appy).	Rule
7	∀x SmartPhoneTech(x) ⇒ Business(x)	business(X) :- smartPhoneTech(X).	Rule

Overall Summary of Facts and Rules:

Facts:

1. competitor(sumsum,appy).
2. company(sumsum).
3. company(appy).
4. developed(sumsum, galatica-s3).
5. smartPhoneTech(galatica-s3).
6. boss(stevey).

Rules:

1. steal(stevey, X, sumsum) :- smartphonetech(X), developed(sumsum,X).
2. unethical(X,Y,Z) :- boss(X), business(Y), rival(Z), steal(X,Y,Z).
3. rival(X) :- competitor(X,appy).
4. business(X) :- smartPhoneTech(X).

Exercise 1

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

Facts:

1. competitor(sumsum,appy).
2. company(sumsum).
3. company(appy).
4. developed(sumum,galatica-s3).
5. smartphonetech(galatica-s3).
6. boss(stevey).

Rules:

1. steal(stevey,X,sumsum) :- smartphonetech(X), developed(sumsum,X).
2. business(X) :- smartphonetech(X).
3. rival(X) :- competitor(X,appy).
4. unethical(X) :- boss(X), business(Y), rival(Z), company(Z), steal(X,Y,Z).

Trace:

```
[trace]  ?- unethical(stevey).
[   Call: (10) unethical(stevey) ? creep
[   Call: (11) boss(stevey) ? creep
[   Exit: (11) boss(stevey) ? creep
[   Call: (11) business(_12800) ? creep
[   Call: (12) smartphonetech(_12844) ? creep
[   Exit: (12) smartphonetech(galatica-s3) ? creep
[   Exit: (11) business(galatica-s3) ? creep
[   Call: (11) rival(_12982) ? creep
[   Call: (12) competitor(_13026, appy) ? creep
[   Exit: (12) competitor(sumsum, appy) ? creep
[   Exit: (11) rival(sumsum) ? creep
[   Call: (11) company(sumsum) ? creep
[   Exit: (11) company(sumsum) ? creep
[   Call: (11) steal(stevey, galatica-s3, sumsum) ? creep
[   Call: (12) smartphonetech(galatica-s3) ? creep
[   Exit: (12) smartphonetech(galatica-s3) ? creep
[   Call: (12) developed(sumsum, galatica-s3) ? creep
[   Exit: (12) developed(sumsum, galatica-s3) ? creep
[   Exit: (11) steal(stevey, galatica-s3, sumsum) ? creep
[   Exit: (10) unethical(stevey) ? creep
true.

[trace]  ?-
```

Figure 1 trace, unethical(stevey) result

To further prove that Stevey is unethical, we can represent the entire process in the form of a ***and-or proof tree.*** The *'OR' node* comes from the choice of a rule or fact to match to a goal. The *'AND' node* comes from the multiple antecedents of a rule (all of which must be proved). (Note: This is just a way of visualizing the search process).

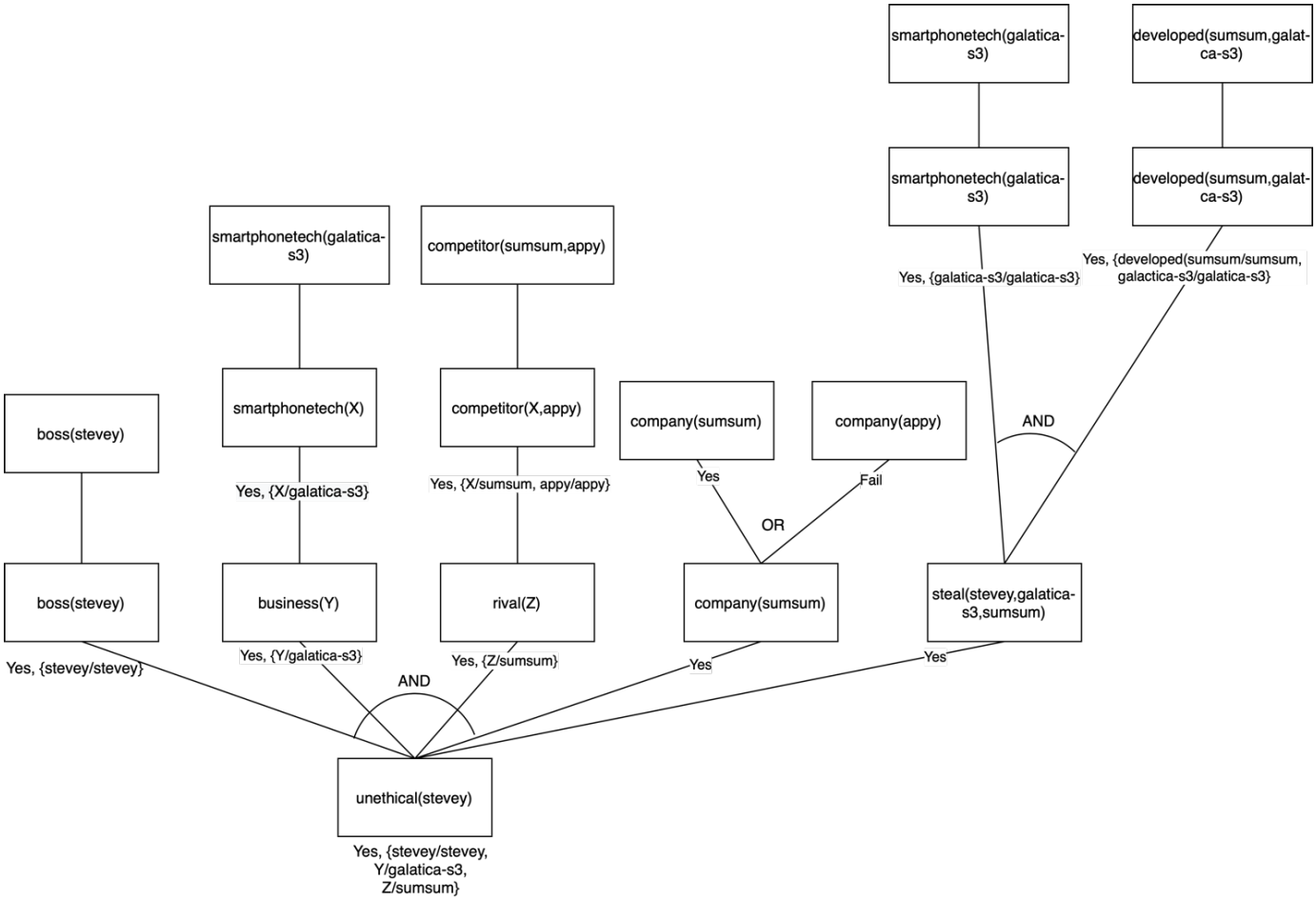

Figure 2 And-Or Proof Tree (with Backtracking)

Exercise 2

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.

Facts:

1. male(charles).
2. male(andrew).
3. male(edward).
4. female(ann).
5. female(elizabeth).
6. queen(elizabeth).

Relationships:

1. mother(elizabeth, charles).
2. mother(elizabeth, ann).
3. mother(elizabeth, andrew).
4. mother(elizabeth, edward).
5. older(charles, ann).
6. older(charles, andrew).
7. older(charles, edward).
8. older(ann, andrew).
9. older(ann, edward).
10. older(andrew, edward).

Rules:

1. older_result(X,Y) :- male(X), male(Y), older(X,Y).
2. older_result(X,Y) :- male(X), female(Y), Y \= elizabeth.
3. older_result(X,Y) :- female(X), female(Y),older(X,Y).

Insertion Sort Algorithm:

1. succession([A|B], Sorted) :- succession(B, SortedTail), insert(A, SortedTail, Sorted). succession([],[]).
2. insert(A, [B|C], [B|D]) :- not(older_result(A,B)),!, insert(A, C, D).
3. insert(A, C, [A|C]).

Returns a list of sorted successions:

1. successionList(X, SuccessionList):- findall(Y, mother(X,Y), Children), succession(Children, SuccessionList).

Trace:

```
[[trace]  ?- successionList(Children, OldSuccessionRank).
[   Call: (10) successionList(_12930, _12932) ? creep
[^  Call: (11) findall(_13366, mother(_12930, _13366), _13426) ? creep
[   Call: (16) mother(_12930, _13366) ? creep
[   Exit: (16) mother(elizabeth, charles) ? creep
[   Redo: (16) mother(_12930, _13366) ? creep
[   Exit: (16) mother(elizabeth, ann) ? creep
[   Redo: (16) mother(_12930, _13366) ? creep
[   Exit: (16) mother(elizabeth, andrew) ? creep
[   Redo: (16) mother(_12930, _13366) ? creep
[   Exit: (16) mother(elizabeth, edward) ? creep
[^  Exit: (11) findall(_13366, user:mother(_12930, _13366), [charles, ann, andrew, edward]) ? creep
[   Call: (11) succession([charles, ann, andrew, edward], _12932) ? creep
[   Call: (12) succession([ann, andrew, edward], _13966) ? creep
[   Call: (13) succession([andrew, edward], _14010) ? creep
[   Call: (14) succession([edward], _14054) ? creep
[   Call: (15) succession([], _14098) ? creep
[   Exit: (15) succession([], []) ? creep
[   Call: (15) insert(edward, [], _14188) ? creep
[   Exit: (15) insert(edward, [], [edward]) ? creep
[   Exit: (14) succession([edward], [edward]) ? creep
[   Call: (14) insert(andrew, [edward], _14326) ? creep
[^  Call: (15) not(older_result(andrew, edward)) ? creep
[   Call: (16) older_result(andrew, edward) ? creep
[   Call: (17) male(andrew) ? creep
[   Exit: (17) male(andrew) ? creep
[   Call: (17) male(edward) ? creep
[   Exit: (17) male(edward) ? creep
[   Call: (17) older(andrew, edward) ? creep
[   Exit: (17) older(andrew, edward) ? creep
[   Exit: (16) older_result(andrew, edward) ? creep
[^  Fail: (15) not(user:older_result(andrew, edward)) ? creep
[   Redo: (14) insert(andrew, [edward], _14828) ? creep
[   Exit: (14) insert(andrew, [edward], [andrew, edward]) ? creep
[   Exit: (13) succession([andrew, edward], [andrew, edward]) ? creep
[   Call: (13) insert(ann, [andrew, edward], _14966) ? creep
[^  Call: (14) not(older_result(ann, andrew)) ? creep
[   Call: (15) older_result(ann, andrew) ? creep
[   Call: (16) male(ann) ? creep
[   Fail: (16) male(ann) ? creep
[   Redo: (15) older_result(ann, andrew) ? creep
[   Call: (16) male(ann) ? creep
[   Fail: (16) male(ann) ? creep
[   Redo: (15) older_result(ann, andrew) ? creep
[   Call: (16) female(ann) ? creep
[   Exit: (16) female(ann) ? creep
[   Call: (16) female(andrew) ? creep
[   Fail: (16) female(andrew) ? creep
[   Fail: (15) older_result(ann, andrew) ? creep
[^  Exit: (14) not(user:older_result(ann, andrew)) ? creep
[   Call: (14) insert(ann, [edward], _14956) ? creep
[^  Call: (15) not(older_result(ann, edward)) ? creep
[   Call: (16) older_result(ann, edward) ? creep
[   Call: (17) male(ann) ? creep
[   Fail: (17) male(ann) ? creep
[   Redo: (16) older_result(ann, edward) ? creep
[   Call: (17) male(ann) ? creep
[   Fail: (17) male(ann) ? creep
[   Redo: (16) older_result(ann, edward) ? creep
[   Call: (17) female(ann) ? creep
[   Exit: (17) female(ann) ? creep
```

Figure 3 Trace of OldRoyalRanks Part 1

```
[   Call: (17) female(edward) ? creep
[   Fail: (17) female(edward) ? creep
[   Fail: (16) older_result(ann, edward) ? creep
[^  Exit: (15) not(user:older_result(ann, edward)) ? creep
[   Call: (15) insert(ann, [], _15634) ? creep
[   Exit: (15) insert(ann, [], [ann]) ? creep
[   Exit: (14) insert(ann, [edward], [edward, ann]) ? creep
[   Exit: (13) insert(ann, [andrew, edward], [andrew, edward, ann]) ? creep
[   Exit: (12) succession([ann, andrew, edward], [andrew, edward, ann]) ? creep
[   Call: (12) insert(charles, [andrew, edward, ann], _12932) ? creep
[^  Call: (13) not(older_result(charles, andrew)) ? creep
[   Call: (14) older_result(charles, andrew) ? creep
[   Call: (15) male(charles) ? creep
[   Exit: (15) male(charles) ? creep
[   Call: (15) male(andrew) ? creep
[   Exit: (15) male(andrew) ? creep
[   Call: (15) older(charles, andrew) ? creep
[   Exit: (15) older(charles, andrew) ? creep
[   Exit: (14) older_result(charles, andrew) ? creep
[^  Fail: (13) not(user:older_result(charles, andrew)) ? creep
[   Redo: (12) insert(charles, [andrew, edward, ann], _12932) ? creep
[   Exit: (12) insert(charles, [andrew, edward, ann], [charles, andrew, edward, ann]) ? creep
[   Exit: (11) succession([charles, ann, andrew, edward], [charles, andrew, edward, ann]) ? creep
[   Exit: (10) successionList(_12930, [charles, andrew, edward, ann]) ? creep
 OldSuccessionRank = [charles, andrew, edward, ann].

[[trace]  ?-
[|
[|
```

Figure 4 Trace of OldRoyalRanks Part 2; OldSuccessionRank = [charles, andrew, edward, ann].

Exercise 2

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.

Modification:

1. From:
    a. older_result(X,Y) :- male(X), male(Y), older(X,Y).
    b. older_result(X,Y) :- male(X), female(Y), Y \= elizabeth.
    c. older_result(X,Y) :- female(X), female(Y),older(X,Y).

    To:
    a. new_result(X,Y) :- older(X,Y).
2. From:
    insert(A, [B|C], [B|D]) :- not(older_result(A,B)),!, insert(A, C, D).

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

Explanation:

Since the new succession ranks does not look at gender as part of the succession criteria, but solely based on age. As such, we can ignore the implied older_rank(X,Y) (*that takes into account of the gender*), and instead, older(X,Y) ⇒ new_result(X,Y).

Next, we apply the new_result(A,B) rule, to the insertion sort algorithm to give us the correct order of the new succession ranks.

Overall Summary (repeat from 2.1 with modifications added):

Facts:

1. male(charles).
2. male(andrew).
3. male(edward).
4. female(ann).
5. female(elizabeth).
6. queen(elizabeth).

Relationships:

1. mother(elizabeth, charles).
2. mother(elizabeth, ann).
3. mother(elizabeth, andrew).
4. mother(elizabeth, edward).
5. older(charles, ann).
6. older(charles, andrew).
7. older(charles, edward).
8. older(ann, andrew).
9. older(ann, edward).
10. older(andrew, edward).

Rules:

1. ***new_result(X,Y) :- older(X,Y).***

Insertion Sort Algorithm:

4. succession([A|B], Sorted) :- succession(B, SortedTail), insert(A, SortedTail, Sorted). succession([],[]).
5. insert(A, [B|C], [B|D]) :- not(***new_result(A,B)***),!, insert(A, C, D).
6. insert(A, C, [A|C]).

Returns a list of sorted successions:

2. successionList(X, SuccessionList):- findall(Y, mother(X,Y), Children), succession(Children, SuccessionList).

Trace:

```
[trace]  ?- successionList(Children, NewSuccessionRank).
   Call: (10) successionList(_12476, _12478) ? creep
^  Call: (11) findall(_12912, mother(_12476, _12912), _12972) ? creep
   Call: (16) mother(_12476, _12912) ? creep
   Exit: (16) mother(elizabeth, charles) ? creep
   Redo: (16) mother(_12476, _12912) ? creep
   Exit: (16) mother(elizabeth, ann) ? creep
   Redo: (16) mother(_12476, _12912) ? creep
   Exit: (16) mother(elizabeth, andrew) ? creep
   Redo: (16) mother(_12476, _12912) ? creep
   Exit: (16) mother(elizabeth, edward) ? creep
^  Exit: (11) findall(_12912, user:mother(_12476, _12912), [charles, ann, andrew, edward]) ? creep
   Call: (11) succession([charles, ann, andrew, edward], _12478) ? creep
   Call: (12) succession([ann, andrew, edward], _13512) ? creep
   Call: (13) succession([andrew, edward], _13556) ? creep
   Call: (14) succession([edward], _13600) ? creep
   Call: (15) succession([], _13644) ? creep
   Exit: (15) succession([], []) ? creep
   Call: (15) insert(edward, [], _13734) ? creep
   Exit: (15) insert(edward, [], [edward]) ? creep
   Exit: (14) succession([edward], [edward]) ? creep
   Call: (14) insert(andrew, [edward], _13872) ? creep
^  Call: (15) not(new_result(andrew, edward)) ? creep
   Call: (16) new_result(andrew, edward) ? creep
   Call: (17) older(andrew, edward) ? creep
   Exit: (17) older(andrew, edward) ? creep
   Exit: (16) new_result(andrew, edward) ? creep
^  Fail: (15) not(user:new_result(andrew, edward)) ? creep
   Redo: (14) insert(andrew, [edward], _14198) ? creep
   Exit: (14) insert(andrew, [edward], [andrew, edward]) ? creep
   Exit: (13) succession([andrew, edward], [andrew, edward]) ? creep
   Call: (13) insert(ann, [andrew, edward], _14336) ? creep
^  Call: (14) not(new_result(ann, andrew)) ? creep
   Call: (15) new_result(ann, andrew) ? creep
   Call: (16) older(ann, andrew) ? creep
   Exit: (16) older(ann, andrew) ? creep
   Exit: (15) new_result(ann, andrew) ? creep
^  Fail: (14) not(user:new_result(ann, andrew)) ? creep
   Redo: (13) insert(ann, [andrew, edward], _14662) ? creep
   Exit: (13) insert(ann, [andrew, edward], [ann, andrew, edward]) ? creep
   Exit: (12) succession([ann, andrew, edward], [ann, andrew, edward]) ? creep
   Call: (12) insert(charles, [ann, andrew, edward], _12478) ? creep
^  Call: (13) not(new_result(charles, ann)) ? creep
   Call: (14) new_result(charles, ann) ? creep
   Call: (15) older(charles, ann) ? creep
   Exit: (15) older(charles, ann) ? creep
   Exit: (14) new_result(charles, ann) ? creep
^  Fail: (13) not(user:new_result(charles, ann)) ? creep
   Redo: (12) insert(charles, [ann, andrew, edward], _12478) ? creep
   Exit: (12) insert(charles, [ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
   Exit: (11) succession([charles, ann, andrew, edward], [charles, ann, andrew, edward]) ? creep
   Exit: (10) successionList(_12476, [charles, ann, andrew, edward]) ? creep
NewSuccessionRank = [charles, ann, andrew, edward].

[trace]  ?-
|
|
|
```

Figure 5 Trace of NewRoyalRanks; NewSuccessionRank =[charles, ann, andrew, edward].