Question 2 - Unifications and resolutions in Prolog **Zafir Khalid - 40152164**

Q1 Done in q1.pl

Which of the following pairs of terms can be unified together? If they can't be unified, please provide the reason for it. If they can be unified successfully, wherever relevant, provide the variable instantiations that lead to successful unification. (Note **=** shows unification)

- likes(jane, X) = likes(X, josh).

FALSE, because of conflicting values of X. X can not equal jane and josh at the same time.

- diSk(27, queens, sgt_pepper) = diSk(A, B, help).

FALSE, because of conflicting values in 3rd atom position. Also since help is not a variable.

- [a,b,c] = [X,Y,Z|T].

TRUE X = a, Y = b, Z = c, T = []

- ancestor(french(jean), B) = ancestor(A, irish(joe)).

TRUE A = french(jean).
B = irish(joe).

- characters(hero(luke), X) = characters(X, villain(vader)).

FALSE, because of conflicting values of X. X can not equal hero(luke) and villain(vader) at the same time.

- f(X, a(b,c)) = f(d, a(Z, c)).

TRUE X = d, Z = b

- s(x, f(x), z) = s(g(y), f(g(b)), y).

FALSE, because of conflicting values in atom positions. For example: $z \neq y$ and $x \neq g(y)$ and $x \neq g(b)$. No variables present here

- vertical(line(point(X,Y), point(X,Z))) = vertical(line(point(1,1),point(1,3))).

TRUE X = 1.
Y = 1.
Z = 3.

- g(Z, f(A, 17, B), A+B, 17) = g(C, f(D, D, E), C, E).

TRUE A=B, B=D, D=E, E=17, C=Z, Z=17+17

- f(c, a(b,c)) = f(Z, a(Z, c)).

FALSE, because of conflicting values of Z. Z can not be equal to c and b at the same time.

Determine the type of each of the following queries (ground/non-ground), and explain what will Prolog respond for each of these queries (write all the steps of unifications and resolutions for each query)?

- ? **building(library, lb)**.

Prolog response:

true.

Type of query:

Ground Query (No variables involved)

Explanation:

Prolog will move sequentially through the knowledge base looking for the query. On line 1 and line 2 prolog encounters building(engineering, ev) and building(business, mb) respectively. Neither unify in this case because the atoms library and lb do not match with the atoms inside these two statements. On line 3 prolog encounters building(library, lb). with which it unifies and hence responds true.

- ? **status(finance, A)**.

Prolog response:

false.

Type of query:

Non-ground Query ("A" is a variable)

Explanation:

status(X, Z) :- department(X, Y), status(Y, Z).
status(finance, A) :- department(finance, Y), status(Y, A).

department(finance, Y)
Y = business

status(Y, A)
status(business, A)
status(business, A) :- department(business, Y'), status(Y', A).

department(business, Y')
FAIL
status(business, A)
FAIL
status(finance, A)
FAIL

- ? **department(civil, Business)**.

Prolog response:

Business = engineering.

Type of query:

Non-ground query ("Business" is a variable)

Explanation:

Unifies with the fact in the KB:

department(civil, engineering).

Therefore, Business = engineering

- ? **faculty(X, civil)**.

Prolog response:

X = jones;
X = james;
X = davis;
false.

Type of query:

Non-ground query ("X" is a variable)

Explanation:

Unifies with the following facts in the KB:

faculty(jones, civil).
faculty(james, civil).
faculty(davis, civil).

faculty(X, Y) :- department(Z, Y), faculty(X, Z).
faculty(X,civil) :- department(Z, civil), faculty(X, civil)
department(Z, civil)
FAIL
faculty(X, civil)
FAIL

Therefore, no further outputs

- ? **faculty(smith, X)**.

Prolog response:

X = electrical;
X = computer;
X = engineering;
false.

Type of query:

Non-ground query ("X" is a variable)

Explanation:

Unifies with the following facts in the KB:

faculty(smith, electrical).
faculty(smith, computer).

faculty(X, Y) :- department(Z, Y), faculty(X, Z).
faculty(smith, X) :- department(Z, X), faculty(smith, Z)
department(Z, X)
Z = electrical
X = engineering
faculty(smith, Z)
faculty(smith, electrical)
Unifies with fact in the KB
Therefore, faculty(smith, engineering). The remaining attempts FAIL

- ? **department(X, Y)**.

Prolog response:

X = electrical,
Y = engineering ;
X = civil,
Y = engineering ;
X = finance,
Y = business ;
X = ibm-exams,
Y = lb.

Type of query:

Non-ground query ("X" and "Y" are variables)

Explanation:

Unifies with the following facts in the KB:

department(electrical, engineering).
department(civil, engineering).
department(finance, business).
department(ibm-exams, lb).

- ? **faculty(X, civil), department(civil, Y).**

Prolog response:

X = jones,
Y = engineering ;
X = james,
Y = engineering ;
X = davis,
Y = engineering ;
false.

Type of query:

Non-ground query ("X" and "Y" are variables)

Explanation:

faculty(X, civil)
X=jones
department(civil, Y)
Y=engineering
X and Y outputted
faculty(X, civil)
X=james
department(civil, Y)
Y=engineering
X and Y outputted
faculty(X, civil)
X=davis
department(civil, Y)
Y=engineering
X and Y outputted

Prolog attempts unification with the rule:
faculty(X, Y) :- department(Z, Y), faculty(X, Z).
but this fails

- ? **faculty(Smith).**

Prolog response:

Smith = smith ;
Smith = walsh ;
Smith = smith ;
Smith = jones ;
Smith = james ;
Smith = davis ;
Smith = smith ;
Smith = walsh ;
Smith = jones ;
Smith = james ;
Smith = davis ;
false.

Type of query:

Non-ground query ("Smith" is a variable)

Explanation:

faculty(X) :- faculty(X, _).
faculty(Smith) :- faculty(Smith, _).
faculty(Smith, _) unifies with : faculty(smith, electrical).
faculty(walsh, electrical).
faculty(smith, computer).
faculty(jones, civil).
faculty(james, civil).
faculty(davis, civil).

Following this the recursive rule is used:
faculty(X, Y) :- department(Z, Y), faculty(X, Z).
faculty(X, _) :- department(Z, _), faculty(X, Z).
department(Z, _)
Z = electrical
faculty(X, electrical)
X = smith
Backtrack
X = walsh
faculty(X, electrical) :- department(Z, electrical), faculty(X, Z)
department(Z, electrical)
FAIL

Process repeated through backtracking
Through backtracking more values of Z are found and the same process is used

- ? **building**(_, **X**).

Prolog response:

X = ev ;
X = mb ;
X = lb ;
X = h ;
X = fg ;
X = ev ;
X = ev ;
X = mb ;
false.

Type of query:

Non-ground query ("X" is a variable)

Explanation:

Unifies with the following facts in the KB:

building(engineering, ev).
building(business, mb).
building(library, lb).
building(classes, h).
building(hr, fg).

building(X, Y) :- department(X, Z), building(Z, Y).
building(_, X) :- department(_, Z), building(Z, X).
department(_, Z)
Z = engineering
building(engineering, X)
X=ev
Therefore, building(_, ev).
X outputted

From here there is backtracking through building(engineering, X) by using the recursive rule to find more unifications. Following this there is further backtracking in department(_, Z). The same process is repeated from the backtracking sequences.

- ? **status(X, accredited), building(X, Y).**

Prolog response:

X = engineering,
Y = ev ;
X = electrical,
Y = ev ;
X = civil,
Y = ev ;
false.

Type of query:

Non-ground query ("X" and "Y" are variables)

Explanation:

Unifies with the following facts in the KB:

status(engineering, accredited).
building(engineering, ev).

X = engineering
Y = evs

Back track call to building(X, Y) :- department(X, Z), building(Z, Y).

department(engineering, Z)
FAIL.

Back track call to status(X, Z) :- department(X, Y), status(Y, Z).

Z = accredited
department(X, Y).
X = electrical
Y = engineering

Recursive call: status(engineering, accredited)
Base case matched in the KB

Therefore output:

X = electrical
Y = ev

A similar process is repeated using these rules to obtain the following 2 results:

X = civil
Y = ev

- ? **status(_, X), building(X, Y).**

Prolog response:

false.

Type of query:

Non-ground query ("X" and "Y" are variables)

Explanation:

No matches are successful, hence false is outputted.

First unification is with:

status(engineering, accredited).

X = accredited
building(accredited, Y)
* building(X, Y) :- department(X, Z), building(Z, Y). * Called

department(accredited, Z)
FAIL
Therefore building(accredited, Y) fails

Through backtracking the following rule is then used in attempt to find unifications:
status(X, Z) :- department(X, Y), status(Y, Z).

status(_, X) :- department(_, Y), status(Y, X)
department(_, Y)
Y = engineering
status(engineering, X)
X = accredited

building(accredited, Y)
For this the following prolog rule is used:
building(X, Y) :- department(X, Z), building(Z, Y).

building(accredited, Y) :- department(accredited, Z), building(Z, Y).
department(accredited, Z)
FAIL

Therefore building(accredited, Y) FAILS

This process is repeated to search for unifications but this query ultimately finds no unifications and so false. is outputted

- ? **faculty(X), faculty(X, Y), department(Y, _).**

Prolog response:

1

X = smith,
Y = electrical ;
X = walsh,
Y = electrical ;
X = smith,
Y = electrical ;
X = jones,
Y = civil ;
X = james,
Y = civil ;

2

X = davis,
Y = civil ;
X = smith,
Y = electrical ;
X = walsh,
Y = electrical ;
X = jones,
Y = civil ;
X = james,
Y = civil ;
X = davis,
Y = civil ;
false.

Type of query:

Non-ground query ("X" and "Y" are variables)

Explanation:

Prolog rules used in this query:

faculty(X) :- faculty(X, _).
faculty(X, Y) :- department(Z, Y), faculty(X, Z).

faculty(X)
faculty(X, _)
X = smith

faculty(smith, Y)
Unifies with fact in the prolog KB
Y = electrical

department(electrical, _)
department(electrical, engineering)

Therefore
X = smith
Y = electrical

Back track call to faculty(smith, Y)
Unifies with fact in the prolog KB
Y = computer

department(computer, _)
FAIL

Backtrack to faculty(smith, Y)
Prolog rule used: faculty(X, Y) :- department(Z, Y), faculty(X, Z).
faculty(smith, Y) :- department(Z, Y), faculty(X, Z)
department(Z, Y)
Z = electrical Y = engineering

faculty(smith, electrical)
Unifies in the KB

department(engineering, _)
FAIL

Backtrack to faculty(smith, electrical)
Prolog rule used: faculty(X, Y) :- department(Z, Y), faculty(X, Z).
faculty(smith, electrical) :- department(Z, electrical), faculty(X, Z)
department(Z, electrical)
FAIL

This process goes on for all remaining possible X and Y values where each one is compared and then backtracked to cycle through another set of values and checked

- ? **faculty(X), faculty(X, Y), !, department(Y, Z).**

Prolog response:

X = smith,
Y = electrical,
Z = engineering.

Type of query:

Non-ground query ("X", "Y" and "Z" are variables)

Explanation:

faculty(X) :- faculty(X,_)
faculty(X,_)
X = smith

faculty(X,Y)
faculty(smith, Y)
Y = electrical

department(Y, Z)
department(electrical, Z)
Z = engineering

No backtracking because of !

- ? **faculty(X)**, !, **faculty(X, _)**.

Prolog response:

X = smith ;
X = smith ;
X = smith ;
false.

Type of query:

Non-ground query ("X" is a variable)

Explanation:

faculty(X)
faculty(X) :- faculty(X, _).
X = smith

faculty(smith, _)
faculty(smith, electrical)
Direct unification from KB
X = smith outputted

Backtrack to faculty(smith, _)
faculty(smith, _)
faculty(smith, computer)
Direct unification from KB
X = smith outputted

Backtrack to faculty(smith, _)
faculty(smith, _)
faculty(X, Y) :- department(Z, Y), faculty(X, Z).
faculty(smith, _) :- department(Z, _), faculty(smith, Z).

department(Z, _)
Z = electrical

faculty(smith, Z)
faculty(smith, electrical)
Direct unification from KB
X = smith outputted

Backtrack to faculty(smith, Z)
faculty(X, Y) :- department(Z, Y), faculty(X, Z).
faculty(smith, electrical) :- department(Z, electrical), faculty(smith, electrical)

department(Z, electrical)
FAIL

This process is repeated for all possible Z values in department(Z, _)
Following this there is no further backtracking to faculty(X) because of the !

- ? **department(X, _),\+ faculty(_, X).**

Prolog response:

X = finance ;
X = ibm-exams.

Type of query:

Non-ground query ("X" is a variable)

Explanation:

department(X,_)
X = electrical

faculty(_, X)
faculty(_, electrical)
Unifies with faculty(smith, electrical)

backtrack to department(X,_)
X = civil
faculty(_, X)
faculty(_, civil)
Unifies with faculty(jones, civil)

backtrack to department(X,_)
X = finance
faculty(_, X)
faculty(_, finance)
faculty(X, Y) :- department(Z, Y), faculty(X, Z).
faculty(_, finance) :- department(Z, finance), faculty(_, Z)

department(Z, finance)
FAIL

faculty(_, finance)
FAIL

Therefore, X = finance is outputted

backtrack to department(X,_)
X = ibm-exams
faculty(_, ibm-exams)
faculty(_, ibm-exams)
faculty(X, Y) :- department(Z, Y), faculty(X, Z).
faculty(_, ibm-exams) :- department(Z, ibm-exams), faculty(_, Z)

department(Z, ibm-exams)
FAIL
faculty(_, ibm-exams)
FAIL

Therefore, X = ibm-exams is outputted
\+ is the not provable operator and succeeds if the arguments passed are not provable

- ? $\mathbf{exists(P), dateofbirth(P, date(\_, \_, Y)), Y < 1963, salary(P, Salary), Salary < 15000}$.

Prolog response:

P = person(jack, fox, date(27, may, 1940), unemployed),
Y = 1940,
Salary = 0 ;
P = person(lily, armstrong, date(29, may, 1961), unemployed),
Y = 1961,
Salary = 0 ;
P = person(ann, cohen, date(29, may, 1961), unemployed),
Y = 1961,
Salary = 0 ;
P = person(anny, oliver, date(9, may, 1961), unemployed),
Y = 1961,
Salary = 0 ;
P = person(jane, fox, date(9, aug, 1941), works(ntu, 13050)),
Y = 1941,
Salary = 13050 ;
false.

Explanation:

exists(P) -> husband(P) -> family(P,_,_)
At first the following order is ran through prolog. Each husband in the knowledge base is unified with husband(P) via family(P, _, _). This inturn unifies with exists(P). P stores information about a particular person. Following this, dateofbirth(P, date(_, _, Y)) is used to obtain the birth year of person P. Once the year if obtained it is compared to see if it is less than 1963 and if so the salary of person P is obtained and compared to check if it is less than 15,000. If all these cases pass then prolog will output that particular person.

Since prolog moves sequentially through the knowledge base the first candidate it picks up is "John Cohen". This person and their details are stored in the variable P. His year of birth is stored in Y and then compared to check if it is less than 1963. Since it is greater than 1963 this query fails for this particular person.

Prolog keeps moving through the people in the knowledge base by backtracking through families and then cycling through every husband and unifying with exists(P). "John Cohen" is one person in the knowledge base prolog eventually reaches. In this case Y < 1963 is true but the salary stored in Salary is > 15000 and so this query also fails

Prolog keeps cycling through all the husbands and eventually finds a match that works. "Jack Fox" is stored in P. He has a birth year (Y) of 1940 and is unemployed meaning Salary is 0. This satisfies all the conditions and so the query succeeds.

Once all the husbands are completed, prolog back tracks and reaches:
exists(P) -> wife(P) -> family(_, P, _)
Now it cycles through all the wives in the knowledge base repeating the same process mentioned above. 4 total matches are found with the wives in the knowledge base.
(Mentioned above). Lastly, prolog backtracks again but this time cycling through the children in the knowledge base. exists(P) -> child(P) -> family(_, _, Children), member(P, Children). The same process is repeated. Once completed, the search is concluded.

- ? exists(P), dateofbirth(P, date(_, _, Y)), !, Y < 1998, salary(P, Salary), Salary < 20000.

Prolog response:

P = person(john, cohen, date(17, may, 1990), unemployed),
Y = 1990,
Salary = 0.

Explanation:

The presence of "!" ensures there is no backtracking for this query.

exists(P) -> husband(P) -> family(P,_,_)

In this case the query is successful using the first fact in the knowledge base. In addition to this because there is no backtracking prolog does not check any further past the first successful unificaiton

// First unification
P = person(john, cohen, date(17, may, 1990), unemployed)
Y = 1990
Salary = 0

All the conditions are satisfied (ie: 1990<1998 and 0 < 20000).
Therefore these values are outputted.

Since there is no back tracking (due to !) prolog does not search for any more unifications in the knowledge base.

- ? wife(person(GivenName, FamilyName, _, works(_, _))).

Prolog response:

GivenName = grace,
FamilyName = baily ;
GivenName = grace,
FamilyName = baily ;
GivenName = grace,
FamilyName = fox ;
GivenName = jane,
FamilyName = fox.

Explanation:

The following query returns all working wives in the knowledge base.

wife(person) -> family(_, person, _)

Only working wives will be unified because of the works(_,_) distinction. It eliminates the unemployed fact. Prolog moves sequentially through the knowledge base and the query only succeeds for the following unifications. No other unifications are made because the of the employed distinction – this way it does not unify GivenName and FamilyName with anyone who is unemployed.

//Unifications

GivenName = grace,
FamilyName = baily ;
GivenName = grace,
FamilyName = baily ;
GivenName = grace,
FamilyName = fox ;
GivenName = jane,
FamilyName = fox.

- ? child(X), dateofbirth(X, date(_, _,1983)).

Prolog response:

X = person(louie, baily, date(25, may, 1983), unemployed) ;
X = person(louie, baily, date(25, may, 1983), unemployed) ;
X = person(pat, cohen, date(5, may, 1983), works(bcd, 15200)) ;
X = person(jim, cohen, date(5, may, 1983), works(bcd, 15200)) ;
X = person(jimey, oliver, date(5, may, 1983), unemployed) ;

Explanation:

The following query returns all children born in 1983.

child(X) => family(_, _, Children), member(X, Children).
dateofbirth(X, date(_,_,1983)).

As prolog moves sequentially through the knowledge base it encounters children from various years. X is unified to every child in the knowledge base but the latter statement dateofbirth(X, date(_,_,1983)) only succeeds for a select few that are mentioned above. Below are all the unifications that are made.

//Unifications
X = person(louie, baily, date(25, may, 1983), unemployed) ;
X = person(louie, baily, date(25, may, 1983), unemployed) ;
X = person(louie, fox, date(5, may, 1993), unemployed) ;
X = person(pat, cohen, date(5, may, 1983), works(bcd, 15200)) ;
X = person(jim, cohen, date(5, may, 1983), works(bcd, 15200)) ;
X = person(bob, armstrong, date(6, oct, 1999), unemployed) ;
X = person(sam, armstrong, date(8, oct, 1998), unemployed) ;
X = person(patty, oliver, date(8, may, 1984), unemployed) ;
X = person(jimey, oliver, date(5, may, 1983), unemployed) ;
X = person(andy, fox, date(5, aug, 1967), works(com, 12000)) ;
X = person(kai, fox, date(5, jul, 1969), unemployed).

Question 5 – Lists and Backtracking **Zafir Khalid - 40152164**

Consider the database from the previous question and answer the following

1) Write a prolog rule totalIncome/2 to compute the total income of a family.

```
totalIncome([],0).
totalIncome([Individual|Family], TotalIncome) :-
        salary(Individual, CurrentSalary),
         totalIncome(Family, NextSalary),
        TotalIncome is NextSalary + CurrentSalary.
```

2) Write a prolog query to print total income of each family.

```
?- family(Husband, Wife, Children), totalIncome([Husband, Wife|Children], TotalIncome).
```

3) Write a prolog query to print family details of each family that has income per family member less than 2000.

```
?- family(Husband, Wife, Children), totalIncome([Husband, Wife | Children],
TotalIncome), length([Husband, Wife|Children], NumberOfPeople),
((TotalIncome/NumberOfPeople)<2000).
```

4) Write a prolog query to print family details of each family where children's total income is more than their parents.

```
?- family(Husband, Wife, Children), totalIncome(Children, TotalIncomeChildren),
totalIncome([Husband, Wife], TotalIncomeParents),
TotalIncomeChildren>TotalIncomeParents.
```

Knowledge Base

```
flightPath(lax, nrt, 12, 5439).
flightPath(sin, nrt, 7, 3329).
flightPath(jfk, nrt, 14, 6729).
flightPath(jfk, lax, 6, 2469).
flightPath(cdg, lax, 12, 5656).
flightPath(fco, sin, 12, 6245).
flightPath(cdg, jfk, 9, 3624).
flightPath(fco, jfk, 10, 4266).
flightPath(cdg, fco, 2, 684).
flightPath(lju, fco, 3, 743).
flightPath(lju, cdg, 2, 587).

transferTime(lax, 2).
transferTime(nrt, 1).
transferTime(cdg, 1).
transferTime(jfk, 4).
transferTime(sin, 3).
transferTime(fco, 2).
transferTime(lju, 1).
```

1) connection(Start, Destination) to check whether destination can be reached from the starting airport or not. You should consider direct as well as indirect paths.

```
connection(Start, Destination) :- flightPath(Start, Destination, _, _).
connection(Start, Destination) :- flightPath(Start, X, _, _), connection(X, Destination).
```

2) flightTime(Start, Destination, Time, Path) to compute the flight time for all possible paths.

```
flightTime(Start, Destination, Time, Path) :- flightPath(Start, Destination, Time, _),
append([Start],[Destination],Path).

flightTime(Start, Destination, Time, Path) :-
        flightPath(Start, A, T1, _),
        transferTime(A, T2),
        flightTime(A, Destination, T3, IntermediatePath),
        Time is T1+T2+T3,
        append([Start], IntermediatePath, Path).
```

3) pathLength(Path, Length) to compute the length of a given path (path will be a list).

```
checkPath([]).
checkPath([H|T]) :- (flightPath(H, _, _, _); flightPath(_, H, _, _)), checkPath(T).

pathLength(Path, Length) :- checkPath(Path), length(Path, Length).
```

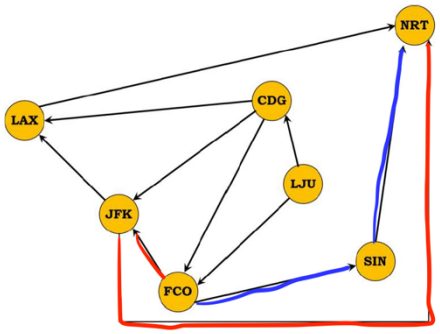
4) shortestPath(Start, Destination) to print the shortest path between two airports.

```
flightDistance(Start, Destination, Distance, Path) :- flightPath(Start, Destination, _, Distance), append([Start],[Destination],Path).

flightDistance(Start, Destination, Distance, Path) :-
        flightPath(Start, A, _, D1),
        flightDistance(A, Destination, D2, IntermediatePath),
        Distance is D1+D2,
        append([Start], IntermediatePath, Path).

shortestPath(Start, Destination) :-
        findall(Distance, flightDistance(Start, Destination, Distance, _), DistanceList),
        min_list(DistanceList, X),
        format('Minimum Distance: ~w', [X]).
```

Testing Section

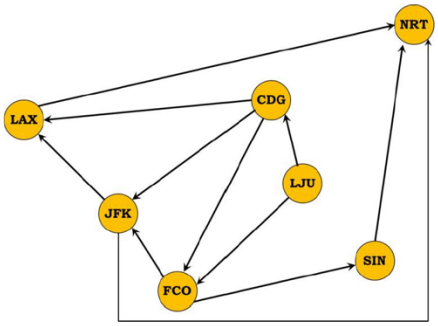
- Rule : connection(Start, Destination)

? – connection(fco, nrt).
true.



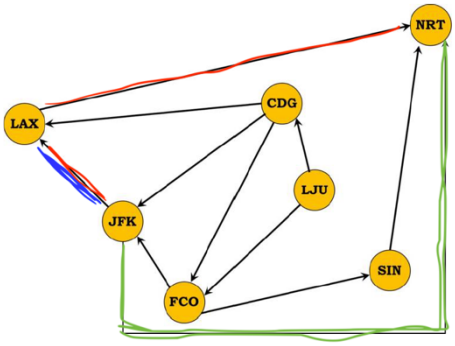
The red and blue lines show that connections indeed do exist from fco to nrt. Hence the response from prolog is correct.

? – connection(nrt, fco).
false.



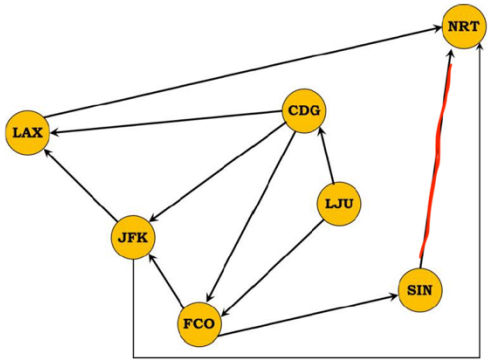
From this diagram we can see that nrt does not have any outgoing flights and so no connections exist taking passengers from nrt to fco. Hence the response from prolog is correct.

? – connection(jfk, X).
X = nrt;
X = lax;
X = nrt;



From the diagram we can see that there are 3 possible connections orignating from jfk airport. The 3 connections are each marked with different colors and are consistent with the results given by prolog. Hence we can conclude that the prolog response is correct

? – connection(sin, X).
X = nrt;



From the diagram we can see that sin only flies to one airport and that is nrt. This is consistent with the prolog response and so we can conclude that the prolog response is correct.

Testing Section

- Rule : flightTime(Start, Destination, Time, Path)

?- flightTime(jfk, nrt, 14, [jfk, nrt]).
true

From the prolog KB we know that the fact flightPath(jfk, nrt, 14, 6729 exists. It shows that the direct flight from jfk to nrt takes 14 hours. As seen by the query above jfk to nrt along the path [jfk, nrt] (direct flight to nrt) takes 14 hours returns true. This is consistent with the fact in our KB and so we can conclude that the prolog response is correct

?- flightTime(lju, fco, 7, [lju, fco]).
false.

From the prolog KB we know that the fact flightPath(lju, fco, 3, 743) exists. It shows that the direct flight from lju to fco takes 3 hours. As seen by the query above it mentions that a direct flight from lju to fco takes 1 hour on the path [lju, fco] (direct flight to fco). We know that it must take 3 hours and not 1 from the KB. Therefore prolog responds false. This answer is consistent with the information in the knowledge base and so we can conclude that the prolog response is correct.

?- flightTime(fco, nrt, Time, [fco, jfk, nrt]).
Time = 28

flightPath(fco, jfk, 10, 4266).
flightPath(jfk, nrt, 14, 6729).
transferTime(jfk, 4).

10+14+4 = 28. So we can conclude that the prolog response is corerct. The sum of the flight time and transit time for a flight from fco to nrt via jfk is 28 hours. This is exactly what prolog returns.

?- flightTime(cdg, jfk, Time, [cdg, jfk]).
Time = 9

flightPath(cdg, jfk, 9, 3624).

The KB shows that a direct flight from cdg to jfk is 9 hours. This is consistent with the prolog response and so we can conclude that the prolog response is correct.

Testing Section

- Rule : pathLength(Path, Length)

?- pathLength([jfk, nrt], 2).
true.

2 is the correct length of the list, so the prolog response is consistent and correct.

?- pathLength([jfk, nrt, sin, dxb], 3).
false.

dxb is not an airport that is included in the graph and so the response is false.

?- pathLength([jfk, nrt, sin], 4).
false.

3 is the correct length of the list, and all the airports mentioned are part of the graph so the prolog response is consistent and correct.

?- pathLength([jfk, nrt, sin, cdg], Length).
Length = 4

4 is the correct length of the list, and all the airports mentioned are part of the graph so the prolog response is consistent and correct.

?- pathLength([jfk], Length).
Length = 1

1 is the correct length of the list and jfk is an airport in the graph, so the prolog response is consistent and correct.

Testing Section

- Rule : shortestPath(Start, Destination)

?- shortestPath(lju, sin).
Minimum Distance: 6988

lju to fco THEN fco to sin
743 + 6245 = 6988

This is the shortest distance as the other path would be [cdg, fco, sin]. That path is longer since it includes the distance of cdg within it. Hence why this is the shortest path and the result is consistent and correct.

?- shortestPath(nrt, sin).
false.

No outbound flights from nrt. Hence the prolog response is correct and consistent in giving false.

?- shortestPath(lju, X).
Minimum Distance: 587

Shortest flight is from lju to cdg which is 587. This response is therefore correct and consistent.

?- shortestPath(jfk, X).
Minimum Distance: 2469

jfk has two out bound flights one to lax (2469) and another to nrt (6729). The shorter of the two is to lax at 2468 km. Hence this result is correct and consistent.