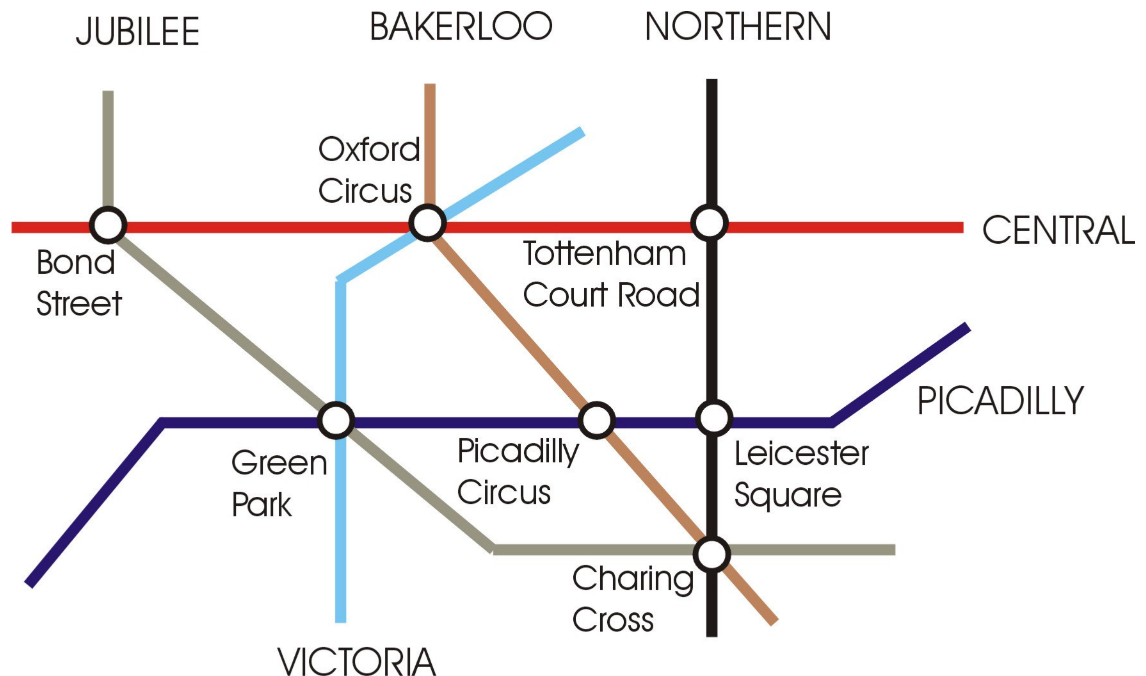
INF2217 – Exercícios 9

Reference

[CS342 PROLOG I and II - Anja.Belz@itri.brighton.ac.uk]

1. Represent the following part of the London Underground by means of the predicate `connected/3`, where the first two arguments are stations and the third argument is the line which connects the stations.



   Now define two stations to be close if they are on the same line, with at most one station in between. Proceed by defining the predicate `reachable/2`, which holds between two stations if they are connected by one or more lines. Finally define the predicate `reachable/3`, where the third argument is the sequence of stations connecting the start station to the end station.

2. A recursive definition of a predicate is a definition which defines a predicate partly in terms of itself. Recursion is the most important programming construct in Prolog. For instance, suppose you wanted to define a predicate `my_number` which is true for the for the following structures (where `s` stands for 'successor'): `0, s(0), s(s(0)), s(s(s(0)))`, etc. In words, `0` is a number, the successor of `0` is a number, etc. The way to go about writing this predicate is as follows. First, begin by writing the following stop clause:

```
my_number(0).
```

Second, complete the definition by adding the following recursive clause:

```
my_number(s(X)) :- my_number(X).
```

- Try running this program. Does it make any difference in what order you put the clauses?
- Define the predicate `sum/3` (the notation `/n` is used to indicate that the predicate has n arguments). For instance, the answer to the query `sum(s(0),s(s(0)),X)` should be `X = s(s(s(0))).` (that is, `1 + 2 = 3`). In other words, the predicate should define the relation of summation between the natural numbers with `0`.

3. Consider the following Prolog knowledge base:

```
p:-q.
r :-p.
q:-r.
p.
```

Query it with `?-p`.

What happens? why? would you have expected this, given the declarative interpretation of the program?

4. Prolog has a built-in predicate '`!`', called the cut. It has no arguments. Its behavior is fundamentally different from ordinary predicates: it has no declarative definition at all; rather it is defined in procedural terms. The cut always succeeds the first time it is called. However, it will make sure that when it is called again during backtracking none of the goals to its left (in the clause in which it occurs), including the head, succeed. For instance, if we have `p:-q,!,r.` then on backtracking, the goals p and q will fail. Now consider the following small Prolog knowledge base:

```
p(X) :- q(X), r(X).
q(a).
r(a).
q(b).
r(X):-s(X).
s(b).
```

If you query this knowledge base with `?- p(X)`, the answers are `X=a` and `X=b`. Modify the knowledge base, inserting the cut somewhere, such that upon querying, the only answer which the system returns is `X=a`.

Answers

1. First define ld connected by following the lines left to right (Northern downwards):

```
ld_connected(bond_street,oxford_circus,central).
ld_connected(oxford_circus,tottenham_court_road,central).
ld_connected(bond_street,green_park,jubilee).
ld_connected(green_park,charing_cross,jubilee).
Etc.
```

The inverse of `ld` connected is `ru` connected That is, we now go right to left and Northern upwards:

```
ru_connected(X,Y,L):-
ld_connected(Y,X,L).
```

Two stations are connected if ld connected or ru connected is true. Note that the predicate ru connected is not strictly necessary to define connected. We could have defined connected directly in terms of ld connected. However, the current definition is probably more transparent.

```
connected(X,Y,L):-
ld_connected(X,Y,L).
connected(X,Y,L):-
ru_connected(X,Y,L).
```

close by is defined in terms of connected:

```
close_by(X,Y):-
connected(X,Y,_L).
close_by(X,Y):-
connected(X,Z,L),
connected(Z,Y,L).
```

Any stations is reachable from any other via at most two intermediate stations. This means that a simple implementation of reachable goes as follows:

```
% no intermediate station:
reachable(X,Y):-
connected(X,Y,_L).

% one intermediate station:
reachable(X,Y):-
connected(X,Z,_L),
connected(Z,Y,_L).

% two intermediate stations:
reachable(X,Y):-
connected(X,A,_L),
connected(A,B,_L),
connected(B,Y,_L).
```

In order to keep track of the route we add a third argument to reachable. Note that reachable/2 and reachable/3, i.e., reachable with respectively 2 and 3 arguments are DIFFERENT predicates.

```
reachable(X,Y,[]):-
connected(X,Y,_L).

reachable(X,Y,[Z]):-
connected(X,Z,_L1),
connected(Z,Y,_L2).

reachable(X,Y,[A,B]):-
connected(X,A,_L1),
connected(A,B,_L2),
connected(B,Y,_L3).
```

2. Yes, the order of the clauses matters.

```
sum(X,0,X).
sum(X,s(Y),s(Z)):-sum(X,Y,Z).
```

3. The program doesn't terminate. This does not follow from the declarative interpretation of the program.

4. Insert the cut after `q(X)` or after `r(X)` in `p(X):- q(X),r(X).`