1.

**?- parent(joost,X).**

**X = sacha ;**

**X = leon.**

After executing the query **parent(joost,X).** Prolog will start looking from the top to the bottom of the list for matching ‘Facts’ with the predicate ‘parent’ which contains two arguments and will match to *parent(joost,...).* With unification prolog will match the terms sacha en leon to X.

And we will get the results **X = sacha** and **X = leon**.

The resolution step in this clause **parent(joost,X)** the literal X will substitute the possible names. Therefore: (joost,sacha),(joost,leon) = (joost,X).

By executing the query **parent(joost,X).** Prolog will start looking from top to bottom of the list if there are any matching 'Facts' with the pridcate of 'parent'. The predicate parent contains two arguments (Y,X). In this query the Y=joost and the X will be matched by unification prolog. As a result we get leon and sacha matched to X.

The resolution step in this clause **parent(joost,X)** the literal X will substitute the possible names. Therefore: (joost,sacha),(joost,leon) = (joost,X).

2.

**?- isChild(X).**

**X = sacha ;**

**X = leon ;**

**X = sacha ;**

**X = leon ;**

**X = sofie ;**

**X = merlijn ;**

**X = fien ;**

**X = joost.**

isChild (X): −parent (Y , X ) . with this rule we define that if there is a clause with a predicate ‘parent’ which has two arguments, the second variable (X) is the child of the given parent (Y). In this clause prolog X substitutes all constants which apply to the condition parent(...,X).

Prolog looks through the list for the predicate ‘parent’ and gives us the second argument (X). That’s why we get duplicates.

isChild (X): −parent (Y , X). This rule defines that if there is a clause with a predicate parent that has 2 arguments(Y,X), the variable named X is the child of variable Y which is the parent in our case. Prolog X substituates all constants which apply for the following condition: parent(..,X).

We get duplicates because Prolog looks through the list for the predicate ‘parent’ and gives us the second argument (X).

3.

**?- brother(X,Z).**

**X = sacha,**

**Z = leon ;**

**X = leon,**

**Z = sacha ;**

**X = sacha,**

**Z = leon ;**

**X = leon,**

**Z = sacha ;**

**X = merlijn,**

**Z = sofie ;**

**X = joost,**

**Z = fien ;**

**false.**

The double entries are because for each predicate ‘parent’ that contains ‘brothers’ we get a result of X being a brother of Z and Z being a brother of X as well. Both X and Z are male in this instance.

For each predicate 'parent' that contains ‘brothers’ we get a result of X being a brother of Z and Z being a brother of X as well. Both X and Z are male in this instance. This is how we get double entries.

**?- sister(X,Z).**

**X = sofie,**

**Z = merlijn ;**

**false.**

4.

**female(fien).**

We do not get a sister(X,Z) X = fien and Z = joost because fien is missing in the database. We can see in the database that Peter has a daughter Fien and a son Joost.

5. And 6. In pl file.

7.

**?- family(sandrine,sofie).**

**False.**

They are not blood related. Sandrine does have two kids from Joost. Using resolution sandrine=X and sofie=Z does not fit in the rules of the family predicate.

By looking at the family rule we can see that if a certain Y is a family of X it needs to atleast fullfill one of the conditions of the family rule. In this case none of the conditions are fullfilled.

8.

**?- family(sandrine,Z).**

**Z = sacha ;**

**Z = leon ;**

**False.**

By looking for Z we find that the variable Z can only be the children of sandrine because she has no other blood relatives.

9.

**?- family(leon,peter).**

**true .**

In this case the grandchild rule is applied. This one is pretty easy as we have already made the rule grandparent in which we find which parent is grandparent of each child. So in the grandchild rule we switch the two variables and we get the child's grandparent.

10.

**?- family(sofie,sacha).**

**true .**

The rule that gives us a result is the cousin(X,Y) rule. In that rule it searches for parents with the same parent thus the grandparent of two different children (X and Y), after that we say that X and Y can not be the same person because they have to be brothers or sisters. In this way we get the children of each parent related to each other but the program also shows us the siblings. So we add \+siblings into the rule and we remove them from the results.

11.pl file

12.

**adjacent(L):-robot(X),link(X,L).**

The adjacent rule gets the current position of the robot (X) and looks for a match (X) in the links (X,L) database if there is one then adjacent is true. That means that X and L are neighbours and a movement from X to L is possible.

If we choose robot(3) and execute the query **adjacent(L).** we get an output of L = 4 and L = 6 and that’s true considering the links and the provided graph.

**move(L):-adjacent(L),retract(robot(\_)),assertz(robot(L)).**

The move rule actually moves the robot to the next position possible considering the links that exist in the current position of the robot.

So if we use robot(1) and execute the query move(L). we will get L=2 and that’s correct as the only move possible from point 1 is to go to point 2.

13.

**suggest(L):-adjacent(L),goal(L).**

If the next possible move (L) has a link to the goal then it suggests the L to reach the goal.

If we add two goals for example 5 and 7, we move the robot to position 6 and execute the query **suggest(L).** we do get both L=5 and L=7 as a possible end goal so this rule works for multiple goals.

14 PL file

15.

16. PL file

17.

18. It will have a infinite loop because 1 ->2->3->1. The link 3 ->1 will be created first and thus suggest(X) will suggest the first path that has a node adjecent. It will not look further down the line.