FAMILY TREE PROBLEM

Submitted By: Muhammad Omair
Reg no: BSCS2019-58
Computer Science (3rd Year)

Submitted to: Sir.Junaid Akhtar
Date: 02 January, 2022

Note: This was a group project of having 7 members in each group. I have successfully done it individually. Thanks to Sir.Junaid Akhtar for having such a wonderful project.

Abstract:

The goal of this project is to combine two languages: Pro-log and Python. For that goal, I created two interfaces: a front-end and a back-end interface. I've worked with pro-log on the back end and Python on the front end. This was done to make it easier for the user to perform any of his or her queries without difficulties. I used this approach to address an issue with the "Khan Family's" family tree. The user may easily type the query and obtain information about the relationships between family members.

Table of Contents:

1. Introduction: _____ 4
2. Goal: _____ 4
3. Pro-log Features: _____ 4
3.1. Facts: _____ 4
3.2. Rules: _____ 5
4. Methodology: _____ 5
4.1. Problem: _____ 5
4.2. Interfaces: _____ 6
4.3. Solution: _____ 7
4.4. Operations: _____ 8
4.5. Tools: _____ 8
4.5.1. SWI-pro-log: _____ 8
4.5.2. Visual Studio Code: _____ 8
4.6. Coding in pro-log: _____ 8
4.7. Coding in python: _____ 8
4.8. Integration and Limitations: _____ 9
5. Conclusion: _____ 9

1. Introduction:

Pro-log is a logic programming language. It also plays a crucial part in artificial intelligence. It's a declarative programming language. In Pro-log, Logic is described as a set of relationships that includes facts and rules. Pattern matching over natural language parse trees is done with Pro-log. We shall tackle the problem of a family tree using this approach. By simply entering the needed information, the user will be able to learn about any of the relationships. The user will type their queries into a Python interface, and Python will retrieve values from the pro-log and return them to the user.

2. Goal:

The first step in this project is to examine the family tree that has been provided. Then, using the three facts provided in the question, we must establish relationships between the nodes of the trees. Following that, we must create functions for the rules that a user might request from the program. To make things easier for the user, we'll write a back-end function in pro-log and a user interface in Python, so that our user can quickly type his query and obtain the desired response.

3. Pro-log Features:

3.1. Facts:

Facts define an object's qualities as well as their relationship. The data bases are formed by the collection of facts, which is then converted into logical format. We just have to deal with three facts in our scenario since we can address each of our needed queries with only three facts. These are the three facts:

- mianbiwi(X,Y).
- parents(X,Y).
- gins(X,Y).

3.2. Rules:

I derive new attributes or relationships of objects from existing ones using rules. The rules are made up of a term called head and a goal called body. Only if the head is correct then is the body correct. Rules variables are shared between the head and the body. Clauses are a term that refers to facts and rules. These rules are used to obtain the results of any query. If my query is valid and the program has a response, the value will be returned; otherwise, the false value will be returned. In our program, I have the following rules:

- pota(X,Y) :-
- dada(X,Y) :-
- nawasa(X,Y) :- etc

4. Methodology:

4.1. Problem:

The "Khan Family" has a family tree that we have. We must create a program that will show us the relationship between any two nodes. If we're looking for A's father, we shouldn't have to look at the entire tree. Our application should produce the desired results by just typing the query into a user-friendly interface. We should create an application that makes the user's life easier. A user may quickly input a simple query to discover the answer to a question about a needed relationship between two members.

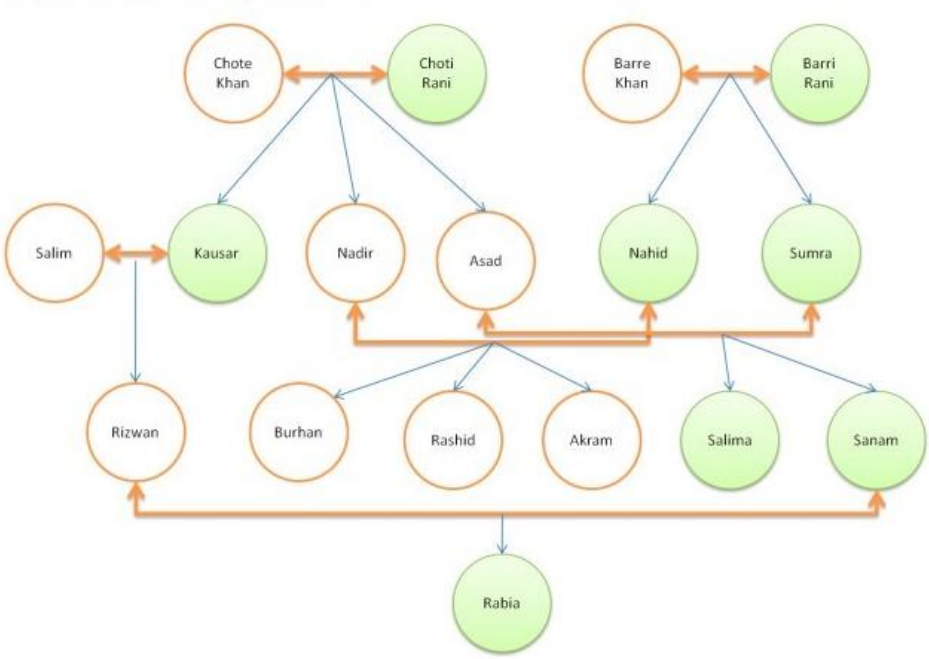

Figure 1: Khan Family Tree

4.2. Interfaces:

In this project, I created two major user interfaces. The first is the back-end, and the second is the front-end interface. In the pro-log programming language, the back-end interface is simply intended to discover the answer to a given problem. For improved user engagement with interfaces, the front-end interface is created in the Python programming language. The major interface is the front-end interface, which is seen in the screenshot below:

```
#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

Enter 1 for Beti_______Enter 2 for Beta
Enter 3 for Dada_______Enter 4 for Nana
Enter 5 for Dadi_______Enter 6 for Nani
Enter 7 for Sala_______Enter 8 for Bahu
Enter 9 for Pota_______Enter 10 for Nawasa
Enter 11 for BaapDada_______Enter 12 for Khala
Enter 13 for Pota_______Enter 0 for Exit

#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

#==> Enter your preference for the type of relationship you want <==#
2
#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#


Khan Family Members:

ChoteKhan, ChotiRani, BarreKhan, BarriRani
Salim, Kausar, Nadir, Asad, Nahid, Sumra
Rizwan, Burhan, Rashid, Akram, Salima, Sanam, Rabia

#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#


Enter person's Name whose Beta is required : nadir

#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
Mr.burhan is the beta of nadir.
#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
Mr.rashid is the beta of nadir.
#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#
Mr.akram is the beta of nadir.
#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#=#

#=#=#=#=#=> Enter C to Continue and S to Stop <=#=#=#=#=#
s


#=#=#=#=#=#=#=#=#===> Thank You  <===#=#=#=#=#=#=#=#=#
```

Figure 2: Interface

4.3. Solution:

I looked at the entire tree. Then, using the data provided, I created relationships between the nodes of the trees. I've created our rules, which we referred to as clauses or functions, based on these facts. These functions will return the result of our query.

To make things easier for the user, I built our back-end code in Pro-Log, which contains all the relations and functions, and my front-end interface in Python, because entering queries in Pro-Log is fairly tough.

4.4. Operations:

A user can perform various operations using our program. Some of them are as follows:

• beta(X,Y) :-

• sala(X,Y) :-

• baapdada(X,Y) :-

• nani(X,Y) :-

• khala(X,Y) :-

• sassur(X,Y) :- etc

A user can receive the value of X by providing the program the value of Y, or the value of Y by giving the program the value of X. Another action that a user may undertake is to examine whether a relationship is genuine or invalid by providing both X and Y values.

4.5. Tools:

I used different programming tools in this project. Details of these tools are given below:

4.5.1. SWI-pro-log:

I've utilized SWI-Pro-log to design the back-end interface in pro-log programming language and code the solution in pro-log programming language.

4.5.2. Visual Studio Code:

I've utilized Visual Studio Code to code the front-end interface in the Python programming language.

4.6. Coding in pro-log:

First, I used pro-log language to solve the provided problem of a family tree sing facts, rules and goals. In this language, I designed the solution using facts, rules, relationships, and goals. I tested the program on several issue goals after completing the solution all of the outcomes were accurate.

4.7. Coding in python:

I had to write the front-end interface in Python when I finished the solution in Pro-log. Python programming was written to improve user engagement. We will take the user's query in the front-end interface and utilize pro-log to discover the solution in the back-end interface. To connect the pro-log and Python program, I utilized the Pyswip package.

To link Python to the prologue, I used the method prolog.consult("FileName"). A previously saved knowledge base in a Prolog file may also be examined and queried. Assuming the filename is "prolog.pl" and the Python is running in the same directory. The function prolog.query(query) executes the user's query.

4.8. Integration and Limitations:

After creating both the back-end and front-end interfaces, I needed to integrate them so that the user could simply utilize the front-end interface while getting the answer from the back-end interface. The user will simply type the query in the front-end interface, and this query will be executed in Python code and delivered to the back-end interface. And the back-end interface will locate the answer to the question and deliver it to the user.

I faced several constraints when designing a solution to a problem.

- I can only utilize three of the facts listed in section 3.1 Facts.

5. Conclusion:

I’ve created the front-end and back-end interfaces in Python and Pro-log, respectively. My final goal was to combine both interfaces so that users could utilize them without having to submit pro-log instructions. I’ve created an interactive user interface with 100% correctness after combining the programs of the pro-log programming language and the python programming language. The user was allowed to input a query to find family member relationships. Following the input of the query, all potential relations from the back-end interface (pro-log program) are returned. This design was extremely simple to use for all types of users.