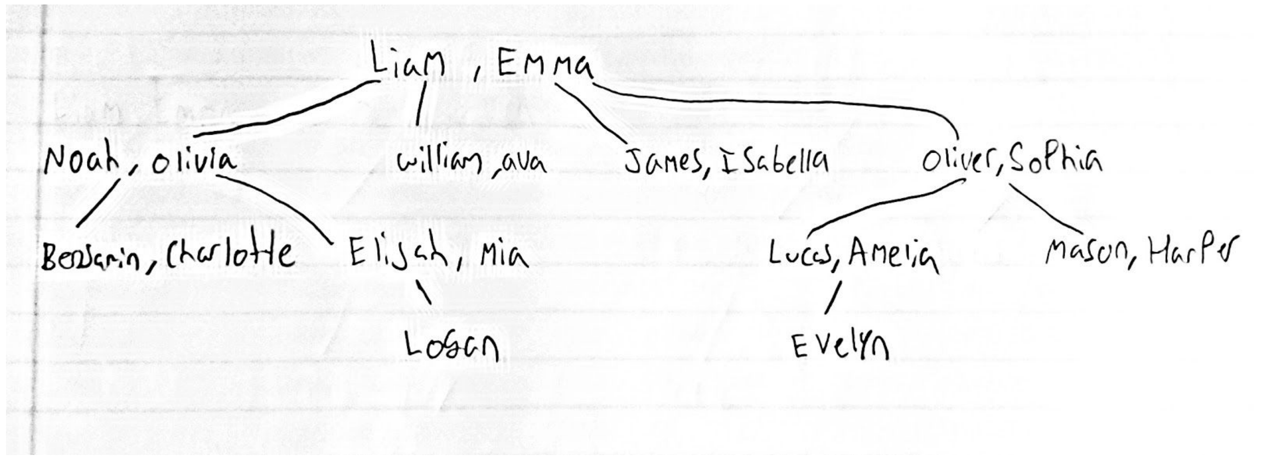
Hunter Hawkins-Stark
CS 470: Artificial Intelligence
Summer 2020
Dr. Soule

Summary of Design and Results

For this project I created a knowledge base to be used in prolog that defines the rules of Genealogy (which attempts to “understand” family relationships).The basis I modeled used the family tree that is represented below.

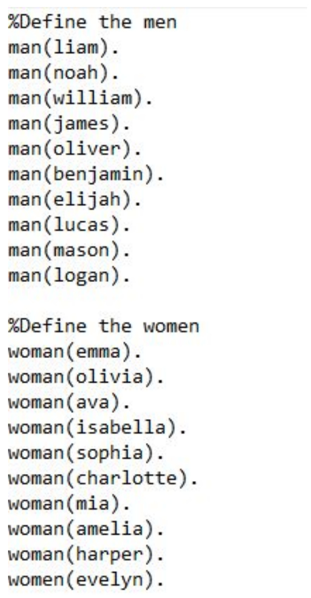


With this family tree in mind, and the logical programming language Prolog. I was able to create rules that defined parents, children, siblings, relatives, descendants, and ancestors. This provides a basic and simple understanding of family relationships that could be expanded upon in the future. As for the results of the system it came back accurate. For every case I have tested it provides the correct output and is generally efficient, although my experience with prolog is extremely limited so I could be incorrect in both aspects given certain inputs.

General Syntax and Example Rules (with there syntax)

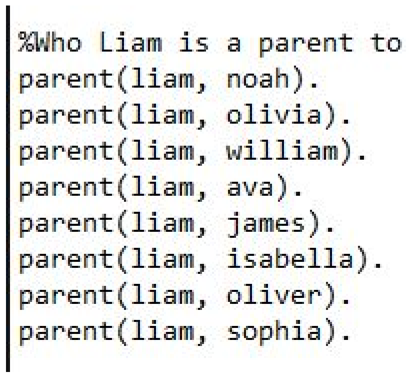
The main design features were rules to define a parent, a child, a sibling, a related family member, a decedent, and an ancestor, for the family tree above. In order to design these rules I had to predefine some facts to the knowledge base. The first facts I added to the knowledge base was 10 women and 10 men in order to start defining relationships. The 10 men and women are defined as follows:

Hunter Hawkins-Stark
CS 470: Artificial Intelligence
Summer 2020
Dr. Soule



```
%Define the men
man(liam).
man(noah).
man(william).
man(james).
man(oliver).
man(benjamin).
man(elijah).
man(lucas).
man(mason).
man(logan).

%Define the women
woman(emma).
woman(olivia).
woman(ava).
woman(isabella).
woman(sophia).
woman(charlotte).
woman(mia).
woman(amelia).
woman(harper).
women(evelyn).
```

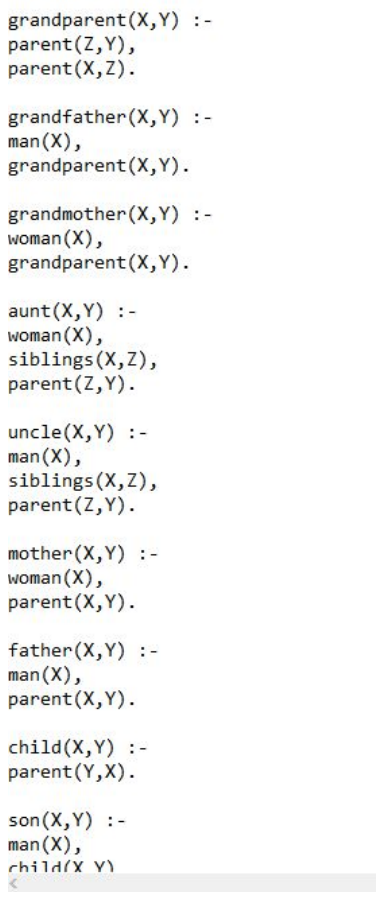
Following adding men and women, I added a baseline of who were parents and who were their children. An example (shown below) is who Liam is a parent to. The parent function takes the first parameter as who is the parent, while the second parameter is the child.



```
%Who Liam is a parent to
parent(liam, noah).
parent(liam, olivia).
parent(liam, william).
parent(liam, ava).
parent(liam, james).
parent(liam, isabella).
parent(liam, oliver).
parent(liam, sophia).
```

As for the rules of the functions, they are pretty straight forward. I tried to address the ones that may be the most complicated first which included the grandparents and aunt and uncles as those seemed to have the most rules, and can involve some digging to figure it out. Some example rules to the functions are shown below.

Hunter Hawkins-Stark
CS 470: Artificial Intelligence
Summer 2020
Dr. Soule



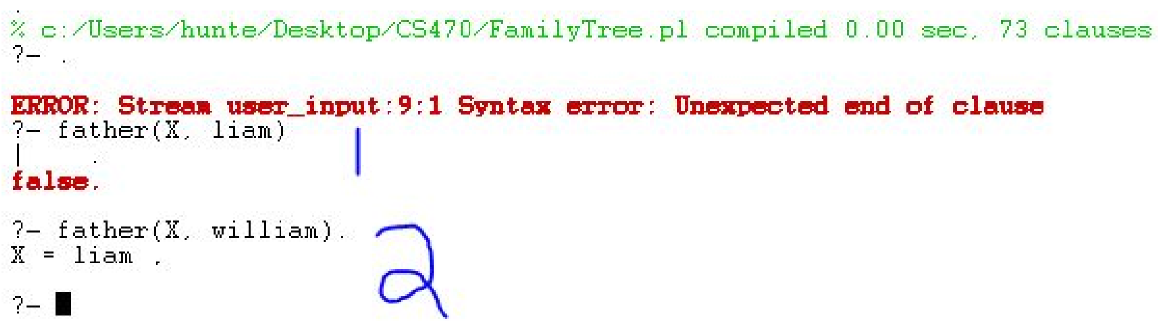
```
grandparent(X,Y) :-
parent(Z,Y),
parent(X,Z).

grandfather(X,Y) :-
man(X),
grandparent(X,Y).

grandmother(X,Y) :-
woman(X),
grandparent(X,Y).

aunt(X,Y) :-
woman(X),
siblings(X,Z),
parent(Z,Y).

uncle(X,Y) :-
man(X),
siblings(X,Z),
parent(Z,Y).

mother(X,Y) :-
woman(X),
parent(X,Y).

father(X,Y) :-
man(X),
parent(X,Y).

child(X,Y) :-
parent(Y,X).

son(X,Y) :-
man(X),
child(X,Y)
```

The general format of the rules are specific rules to closely follow genealogy. This helps make it accurate, easy to change or add to the knowledgebase, and intuitive to anybody that may be looking at the program from an outsider's perspective.

Example Queries

In my first two query examples below I am asking prolog to find fathers of the respective children. In query 1 I asked for the father of Liam, and the system responded appropriately with a false as in the current family tree it doesn't have a mapping of Liam's father. In query two we ask for Williams father which the system correctly returns Liam as Williams father.

Hunter Hawkins-Stark
CS 470: Artificial Intelligence
Summer 2020
Dr. Soule

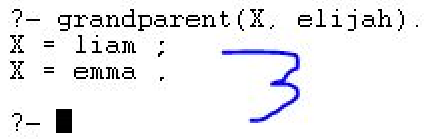


```
% c:/Users/hunte/Desktop/CS470/FamilyTree.pl compiled 0.00 sec, 73 clauses
?- .

ERROR: Stream user_input:9:1 Syntax error: Unexpected end of clause
?- father(X, liam)
|    .
false.

?- father(X, william).
X = liam .

?- 
```

1

2

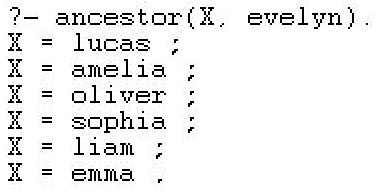
Here we are asking for Elijahs grandparents:



```
?- grandparent(X, elijah).
X = liam ;
X = emma .

?- 
```

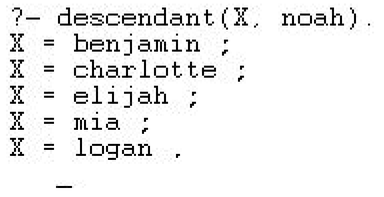
3

Asking for all the ancestors of evelyn:



```
?- ancestor(X, evelyn).
X = lucas ;
X = amelia ;
X = oliver ;
X = sophia ;
X = liam ;
X = emma .
```

Descendent of Noah:



```
?- descendant(X, noah).
X = benjamin ;
X = charlotte ;
X = elijah ;
X = mia ;
X = logan .
```

Overall my Queries seem to work well with my knowledge base. This is my first time using a knowledge base and having prolog consult it, so there could be cases that I'm missing.

Strengths and weaknesses

Some strengths of my design are flexibility, as you can easily add, delete or modify rules in the database. Ease of understanding as all rules and definitions are intuitive and have a consistent naming convention that follows a standard family tree. It is also easy to develop queries for the database due to the intuitive naming of the rules.

Hunter Hawkins-Stark
CS 470: Artificial Intelligence
Summer 2020
Dr. Soule
Some weaknesses of my design could be the accuracy, certain rules may not be 100% accurate and will take time and use to figure out. Along with that it may not be optimized in terms of speed, as there are probably spots in the knowledge base that would slow down the results of the users queries.

Appendix

```
%Define the men
man(liam).
man(noah).
man(william).
man(james).
man(oliver).
man(benjamin).
man(elijah).
man(lucas).
man(mason).
man(logan).

%Define the women
woman(emma).
woman(olivia).
woman(ava).
woman(isabella).
woman(sophia).
woman(charlotte).
woman(mia).
woman(amelia).
woman(harper).
women(evelyn).

%Who Liam is a parent to
parent(liam, noah).
parent(liam, olivia).
parent(liam, william).
parent(liam, ava).
parent(liam, james).
parent(liam, isabella).
parent(liam, oliver).
parent(liam, sophia).

%Who emma is a parent to
```

Hunter Hawkins-Stark
CS 470: Artificial Intelligence
Summer 2020
Dr. Soule

```
parent(emma, noah).
parent(emma, olivia).
parent(emma, william).
parent(emma, ava).
parent(emma, james).
parent(emma, isabella).
parent(emma, oliver).
parent(emma, sophia).

%Who noah is a parent to
parent(noah, benjamin).
parent(noah, charlotte).
parent(noah, elijah).
parent(noah, mia).

%Who Olivia is a parent to
parent(olivia, benjamin).
parent(olivia, charlotte).
parent(olivia, elijah).
parent(olivia, mia).

%william is parent to no one
%ava is parent to no one
%james is parent to no one
%isabella is parent to no one

%Who Oliver is a parent to
parent(oliver, lucas).
parent(oliver, amelia).
parent(oliver, mason).
parent(oliver, harper).

%Who Sophia is a parent to
parent(sophia, lucas).
parent(sophia, amelia).
parent(sophia, mason).
parent(sophia, harper).

%Who Elijah is a parent to
parent(elijah, logan).
```

Hunter Hawkins-Stark
CS 470: Artificial Intelligence
Summer 2020
Dr. Soule

```
%Who Mia is a parent to
parent(mia,logan).

%Who Lucas is a parent to
parent(lucas, evelyn).

%Who Amelia is a parent to
parent(amelia, evelyn).

grandparent(X,Y) :-
parent(Z,Y),
parent(X,Z).

grandfather(X,Y) :-
man(X),
grandparent(X,Y).

grandmother(X,Y) :-
woman(X),
grandparent(X,Y).

aunt(X,Y) :-
woman(X),
siblings(X,Z),
parent(Z,Y).

uncle(X,Y) :-
man(X),
siblings(X,Z),
parent(Z,Y).

mother(X,Y) :-
woman(X),
parent(X,Y).

father(X,Y) :-
man(X),
parent(X,Y).

child(X,Y) :-
parent(Y,X).
```

Hunter Hawkins-Stark
CS 470: Artificial Intelligence
Summer 2020
Dr. Soule

```
son(X,Y) :-
man(X),
child(X,Y).

daughter(X,Y) :-
woman(X),
child(X,Y).

siblings(X,Y) :-
parent(Z, X),
parent(Z, Y),
X\=Y.

brother(X,Y) :-
man(X),
siblings(X,Y).

sister(X,Y) :-
woman(X),
siblings(X,Y).

ancestor(X,Y) :-
descendant(Y,X).

descendant(X,Y) :-
parent(Y,X).

descendant(X,Y) :-
parent(Z,X),
descendant(Z,Y).

% X\=Y is equivalent to not (x = y)
related(X,Y) :-
ancestor(Z,X),
ancestor(Z,Y),
X\=Y.
```