A Report on ChatBot

Members

Antonio Louix Ceriola

Jerald Krister Tabuzo

Percival Jr. Macalintal

Ian Matthew Yanga

Arjay Kyle Rabe

CSINTSY S13 - Group # 2

Table of Contents

I. Introduction

ChatBots are a new area of technology being explored. The older implementations of chatbots were rule-based programs, meaning they only recognized keywords and answered using predefined responses. Chatbots today are integrated with artificial intelligence, machine learning, deep learning, and neural networks; this means that they can utilize machine learning which allows them to analyze and interpret data and answer based on those interpretations in a more human-like response instead of using predetermined answers based on hardcoded rules (Hingrajia, 2023). This project is an implementation of a chatbot that creates a family tree knowledge base, using the user's query information. The programming language used for the family tree chatbot is Python for the command line interface, and PySwip as an extension for the assertion and queries in the knowledge base (Yuce, 2023).

Important: If Receiving Error: **Assertion failed: 0, file /home/swipl/src/swipl-devel/src/pl-fli.c, line 2637**

1. Use this version of pyswip (Requires Git)
   pip install git+https://github.com/yuce/pyswip@master#egg=pyswip

Reference: https://stackoverflow.com/questions/75785959/assertion-failed-when-using-pyswip

2. Transfer .py file to documents directory.

II. Knowledge Base

This section lists the formulas that were encoded into the knowledge base. This includes the numerous rules and the description of the respective formulas. These are represented by using the standard first-order logic notation.

```
male(X), female(Y)
```

X is a male, Y is a female

Equation 1. Male & Female

```
siblings(X, Y) :- parent(Z, X), parent(Z, Y)
```

X and Y are siblings if they share the same parent Z.

Equation 2A. Sibling

```
siblings(X, Y) :- sib(X, Y); sib(Y, X).
```

An alternative definition for the siblings relationship using the sib predicate.

Equation 2B. Sibling

```
parent(X, Y)
```

X is the parent of Y.

Equation 3. Parent

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

X is the grandparent of Y if there exists an intermediate parent Z.

Equation 4. Grandparent

pibling(X, Y) :- parent(Z, X), siblings(Z, Y)

X is a pibling (aunt or uncle) of Y if X is a parent of a sibling of Y.

Equation 5. Pibling (aunt or uncle)

III. Chatbot Implementation

The chatbot is implemented using Python for question prompts and an additional package called "pyswip" for the assertion and query from the knowledge base. The strategy used by our group in designing the chatbot is deviating from exhausting all the possibilities of the knowledge base and using DFS to derive new knowledge from the current knowledge base. This is why we have so few rules in our knowledge base.

A sample scenario for this is the mutual sibling relationship, in a knowledge base, we could create a rule Sibling(X,Y) if Sibling(Z,X) & Sibling(Z,Y), this rule will work in this simple scenario. If A is a sibling of B and B is a sibling of C, then A and C are siblings (Figure 1). However, given a more complex scenario of A being a sibling of B, B being a sibling of C, C is a sibling of D (**Figure 2**). Then **A** and **D** are not siblings, as according to this rule, the only registered sibling of **A** is **B**, and the

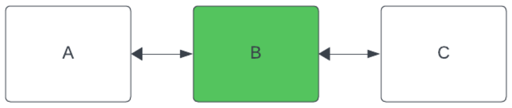


Figure 1. Simple A-B-C Sibling Relationship

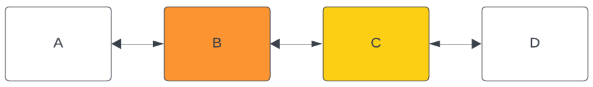


Figure 2. More Complex A-B-C-D Sibling Relationship.

only registered sibling of **D** is **C**, and they both do not have a mutual sibling connection. With this, we could use DFS to find the siblings of A, which is B, then try to find if there is a mutual connection from B to D, and since B and D have mutual siblings C, then A and D are now considered siblings (**Figure 3**).

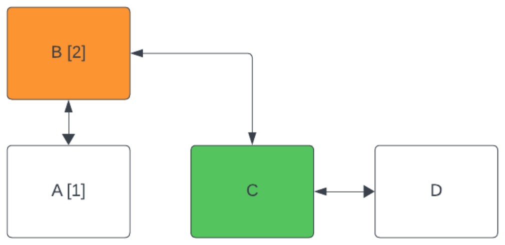


Figure 3. DFS Connection of Siblings A and D.

Another scenario is finding a parent. In a simple scenario, the only rule required is Parent(parent, child) (**Figure 4**). Still, this rule will not work in a scenario where the parent is registered on a sibling (**Figure 5**), and the strategy we used here is first to use DFS to find all siblings of child **D**, then for each sibling, we will use Prolog Query to find the parent of the child. This concept could also be applied to finding the grandparents of the child, where the first step is finding the parents of the child, then finding the siblings of the parents, while using Prolog Query to find the parent of each one. This pattern also applies to finding a pibling and a grandparent.

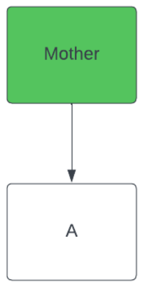


Figure 4. Simple Parent-Child Relationship

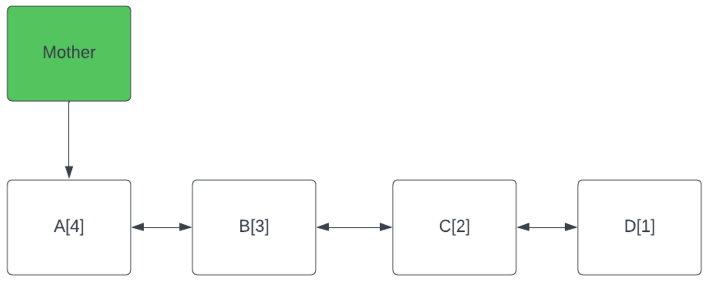


Figure 5. More complex Parent-Child Relationship DFS parent checking starting from Child D

In terms of finding a gender-sensitive relationship, the chatbot asserts a person x as male(x) or female(x), depending on the statement of the user. There is also an error handling involved, where a person who is registered male could not be registered as a female gender, which will be further discussed in the next section. The method of query in terms of gender is by Prolog query, without any other methods used. So given an example of finding the father of X, the parents of child X will first be identified, and then following this, each parent's gender will be checked using Prolog. If a query returns true for male(parent), then it returns that parent's name.

Pseudocode for finding a father of person child

```
findFather():
    parents = set()
    parents = findParents(child) ## DFS Find Children, then PrologQuery parents(X,eachChild)
        for p in parents:
            if isMale(p):
                return p
```

Aside from incorrect user inputs in the chatbot, multiple error-handling features are put in place to prevent invalid assertions into Prolog. Error handling features include the Checking for Impossible Relationships and Gender Sensitive Error Handling.

When checking for impossible relationships, we first designed the chatbot to prevent the user from asserting a person as someone with a relationship with himself, which for example:

"X is the brother of X".

Another assertion covered by the error handling is when the user asserts someone who already has a relationship with X, for example:

"A and B are brothers, therefore, A can not be the parent/aunt/grandparent of B.".

Finally, we also check the number of parents the child has, which in this case, a child should have a maximum of 2 parents and is limited to a single father and mother. Five test cases are covered where

A. **"A is the father of B, and C is the father of D; therefore, B and D can not be siblings, as they both have different fathers."**

B. **"A is the father of B; therefore, C can not be the father of B. As B already has a father."**

C. **"A is the father of B, C is the mother of B, therefore, D can neither be the father and mother of B since B already has both father and mother."**

D. **"P is a parent of A, P2 is a parent of A; therefore A can never have any other parent, father, or mother aside from P and P2."**

E. **"P is a parent of A, M is the mother of A; therefore A can never have any other parent aside from P and P2, and mother aside from P2. In addition to this, P could still be reassigned as a father to A, and vice versa."**

IV. Chatbot Query Results

Here we have a scenario, we have 5 siblings named [a,b,c,d,e] respectively (**Figure 6.1**). On prolog, sibling relationships of [a, b], [b, c], [c, d], [d, e], and [notsiblingswith_a, notsiblingswith_a2] are registered, and there is no connection between a and e. With our design, we used DFS and Prolog to Traverse from A to E. In **Figure 6.2**, we can see that the implementation for this type of scenario is indeed correct.

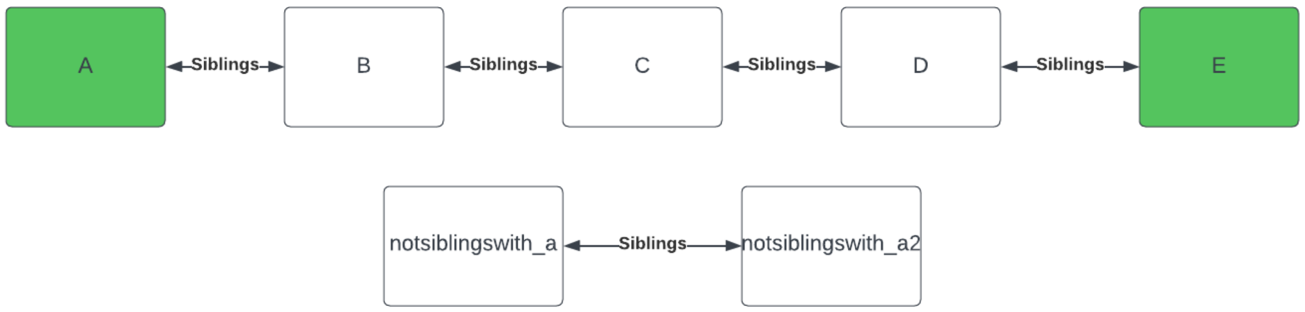
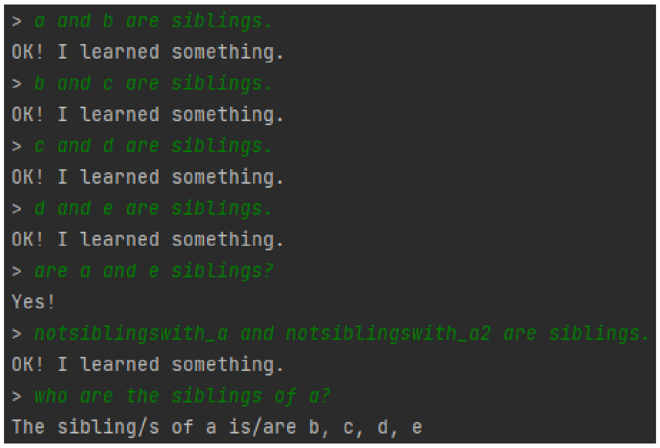


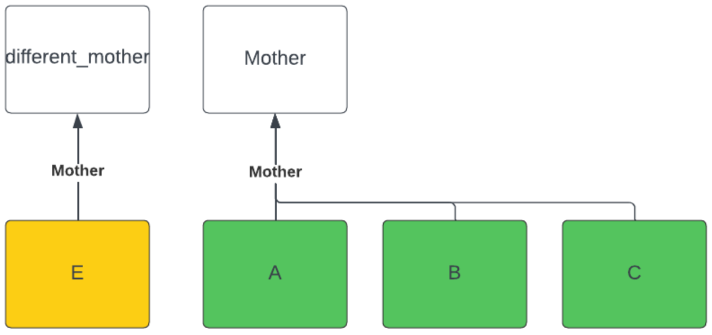
Figure 6.1 Five Siblings Family Tree, No More than 1 Intersibiling Relationship

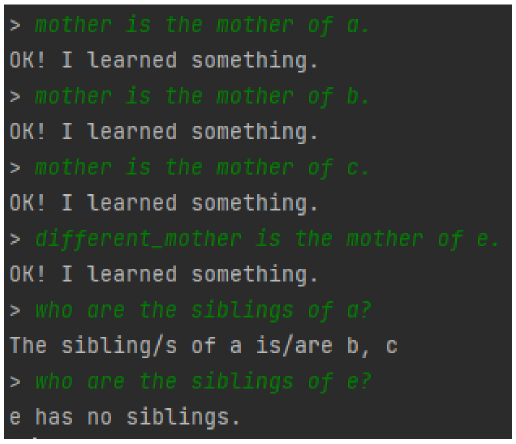


```
> a and b are siblings.
OK! I learned something.
> b and c are siblings.
OK! I learned something.
> c and d are siblings.
OK! I learned something.
> d and e are siblings.
OK! I learned something.
> are a and e siblings?
Yes!
> notsiblingswith_a and notsiblingswith_a2 are siblings.
OK! I learned something.
> who are the siblings of a?
The sibling/s of a is/are b, c, d, e
```

Figure 6.2 Chatbot Testing of Figure 6.1 in the Python Console

Another scenario is where there is a parent, in this case, a mother. Mother has three children [a,b,c]. These children should be siblings as they have the same parent. A buffer prolog assertion of mother(different_mother,e) was made to test if our implementation is indeed correct (**Figure 7.1**). In **Figure 7.2** where we test this case on our chatbot, it passes this test case.





```
> mother is the mother of a.
OK! I learned something.
> mother is the mother of b.
OK! I learned something.
> mother is the mother of c.
OK! I learned something.
> different_mother is the mother of e.
OK! I learned something.
> who are the siblings of a?
The sibling/s of a is/are b, c
> who are the siblings of e?
e has no siblings.
```

Figure 7.1 One Parent- Multiple Children Assertion

Figure 7.2 Chatbot Testing of Figure 7.1 in the Python Console

In the following scenario, a mother is asserted for a and b. Then b, c, and d are siblings, with father being the father of d (**Figure 8.1**). In this case, the parents of any of [a,b,c,d] should be both mother and father, as they are siblings and share the same parent. This validation of this test case is shown below in **Figure 8.2**.

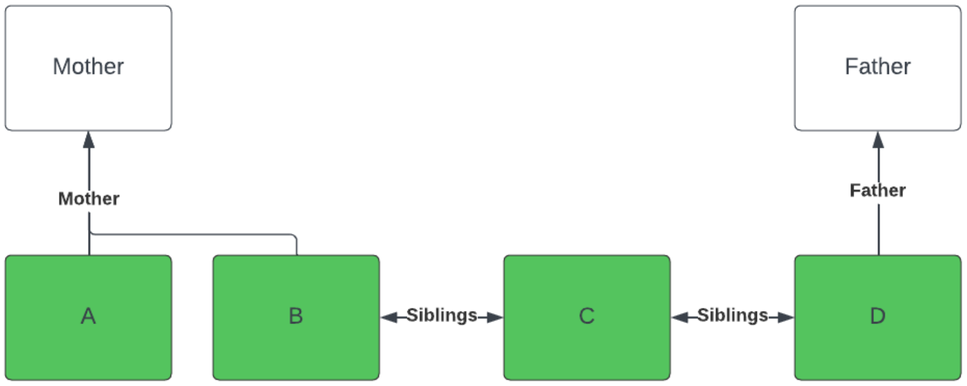
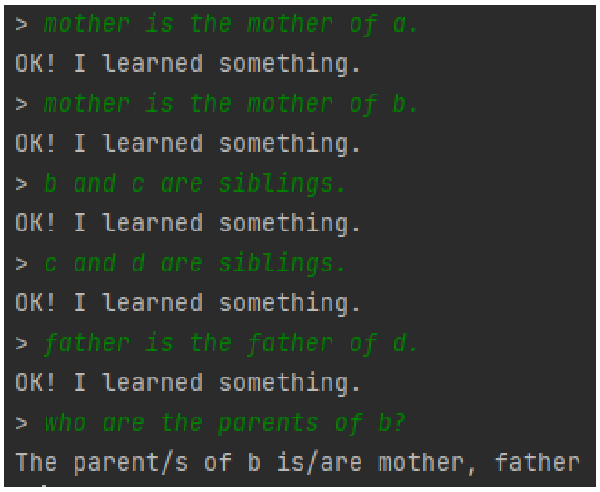


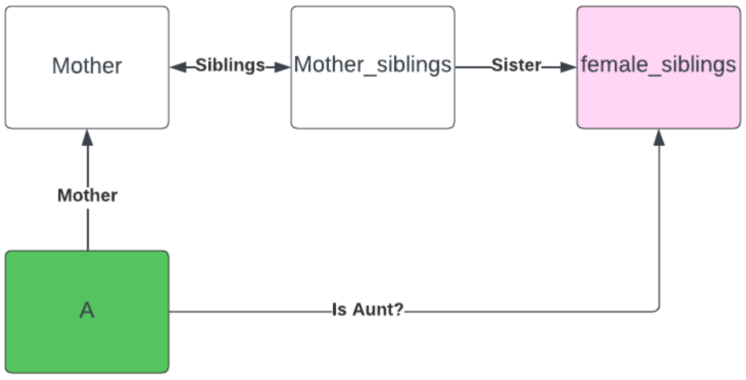
Figure 8.1 Multiple Siblings, Parent Asserted on Two Distinct Siblings

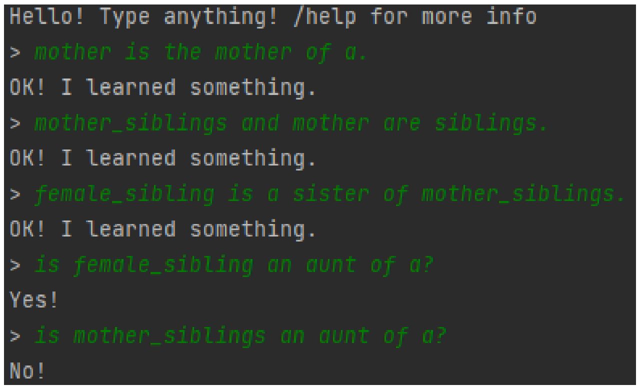


```
> mother is the mother of a.
OK! I learned something.
> mother is the mother of b.
OK! I learned something.
> b and c are siblings.
OK! I learned something.
> c and d are siblings.
OK! I learned something.
> father is the father of d.
OK! I learned something.
> who are the parents of b?
The parent/s of b is/are mother, father
```

Figure 8.2 Chatbot Validation of Scenario in Figure 8.1.

In the next test case, we wanted to determine if x is an aunt of y. In **Figure 9.1**, we have a mother with one child named a. Mother has a sibling named Mother_siblings, and mother_siblings has a sister female_siblings. In this case, only female_siblings is the aunt of a, and mother_siblings was added as a test case. In **Figure 9.2**, we can see that mother_sibligns is not an aunt of a, but female_siblings is, which validates this test case.



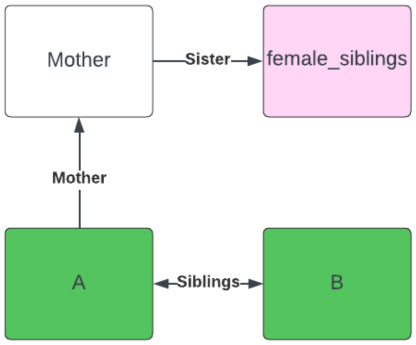


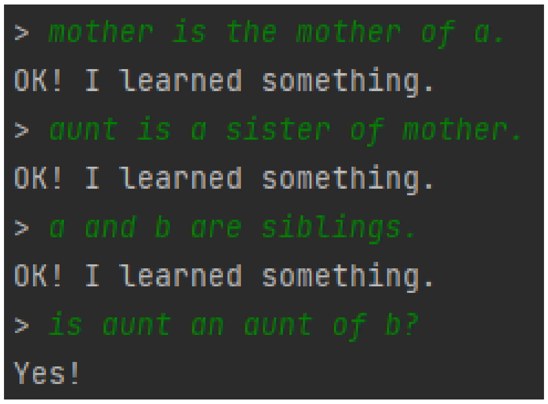
```
Hello! Type anything! /help for more info
> mother is the mother of a.
OK! I learned something.
> mother_siblings and mother are siblings.
OK! I learned something.
> female_sibling is a sister of mother_siblings.
OK! I learned something.
> is female_sibling an aunt of a?
Yes!
> is mother_siblings an aunt of a?
No!
```

Figure 9.1. Finding Aunt through a Parent

Figure 9.2. Chatbot Validation of Figure 9.1

This next test case is a continuation of the previous one. In this scenario, we wanted to verify if a sibling of a, which is b, could confirm if female_siblings are also his/her aunt (**Figure 10.1**). In **Figure 10.2**, we can confirm that the feature extends even to siblings.





```
> mother is the mother of a.
OK! I learned something.
> aunt is a sister of mother.
OK! I learned something.
> a and b are siblings.
OK! I learned something.
> is aunt an aunt of b?
Yes!
```

Figure 10.1 Family Tree Scenario on Finding an Aunt through a Sibling

Figure 10.2 Using the Chatbot to Validate the Situation in Figure 10.1

The next test case is looking for a grandparent, in this case, a grandmother. For this test case, we will use a family tree shown in **Figure 11.1.** Here, we wanted to see if B, who is a sibling of A, could find a grandparent given that B only has a sibling connection, and **Figure 11.2**, we can see that b was able to confirm mothers_mother as his/her grandmother.

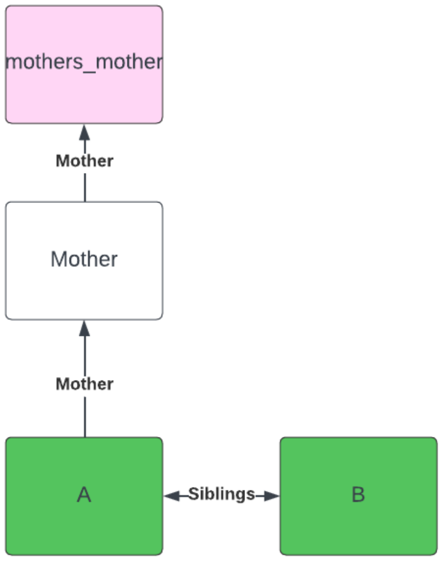
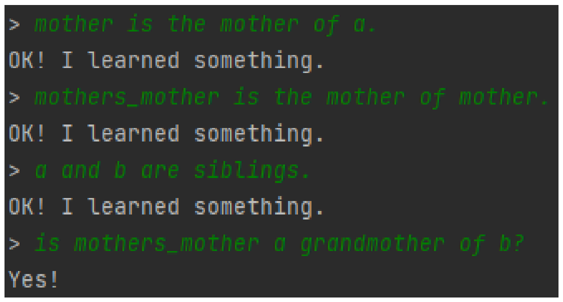


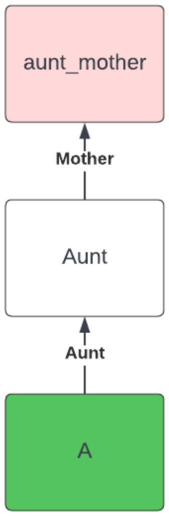
Figure 11.1 Family Tree Scenario for Tracing a Grandmother

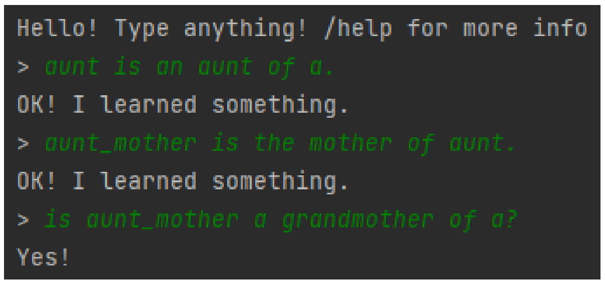


```
> mother is the mother of a.
OK! I learned something.
> mothers_mother is the mother of mother.
OK! I learned something.
> a and b are siblings.
OK! I learned something.
> is mothers_mother a grandmother of b?
Yes!
```

Figure 11.2 Chatbot Test Case Validation

In the following test case, we want to see if we can trace a grandparent through a pibling(also known as a parent's sibling). Here we only have 3 people, where aunt is the Aunt of a and aunt_mother is the mother of aunt (**Figure 12.1**). In **Figure 12.2**, we can validate that a was able to find a path to aunt_mother.



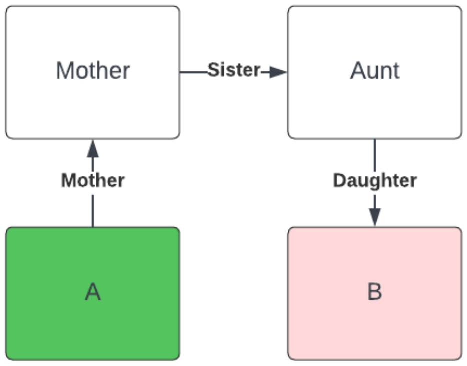


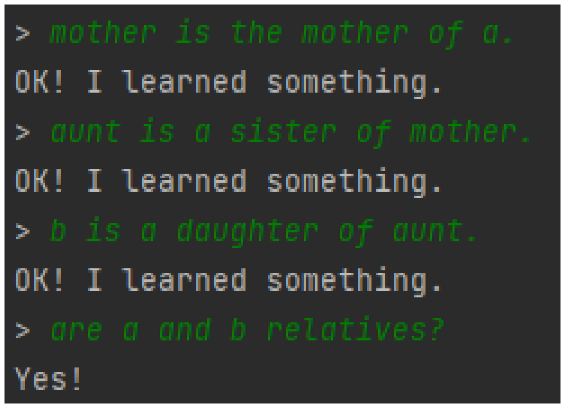
```
Hello! Type anything! /help for more info
> aunt is an aunt of a.
OK! I learned something.
> aunt_mother is the mother of aunt.
OK! I learned something.
> is aunt_mother a grandmother of a?
Yes!
```

Figure 12.1 Family Tree Scenario: Finding a Grandparent.

Figure 12.2 Chatbot Validation of the Scenario in **12.1**

In our final test case, we wanted to test the relatives feature. While a pibling, grandparents, parents, and siblings are considered relatives, we also want to extend the relative feature to cousins (or pibling's child). A family tree is shown in **Figure 13.1.** In **Figure 13.2**, we verified that A and B are relatives (or cousins in this case).





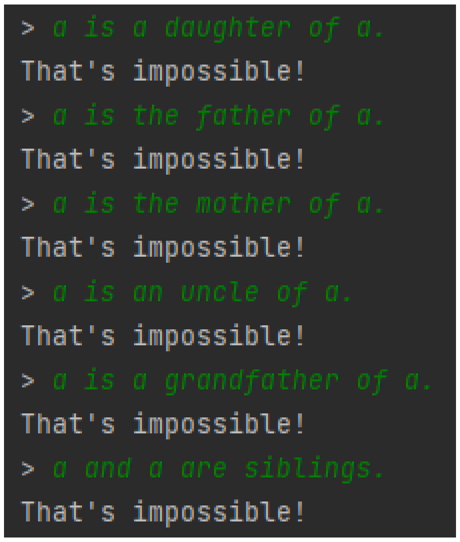
```
> mother is the mother of a.
OK! I learned something.
> aunt is a sister of mother.
OK! I learned something.
> b is a daughter of aunt.
OK! I learned something.
> are a and b relatives?
Yes!
```

Figure 13.1 A family Tree Scenario with a cousin.

Figure 13.2 Chatbot Validation for Figure 13.1

V. Error Handling Results

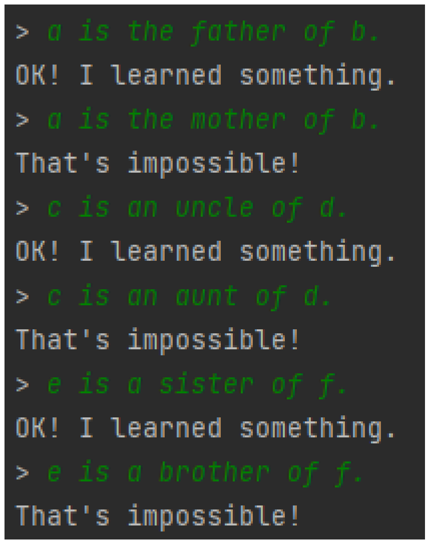
1. For the first error-handling test, we wanted to prevent the user from asserting a self-relationship. In **Figure 14**, we can confirm that it is impossible to create a Self-Assertion Rule.



```
> a is a daughter of a.
That's impossible!
> a is the father of a.
That's impossible!
> a is the mother of a.
That's impossible!
> a is an uncle of a.
That's impossible!
> a is a grandfather of a.
That's impossible!
> a and a are siblings.
That's impossible!
```

Figure 14. Self Assertion Test Case

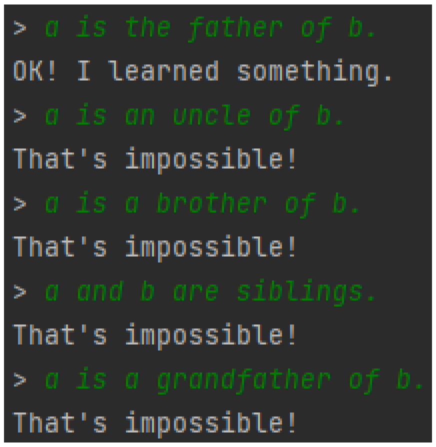
2. In the next error-handling test case, we wanted to prevent the user from creating a single person with two genders. Figure 15 shows that it is impossible to make a female, as a is already a male, along with other test cases.



```
> a is the father of b.
OK! I learned something.
> a is the mother of b.
That's impossible!
> c is an uncle of d.
OK! I learned something.
> c is an aunt of d.
That's impossible!
> e is a sister of f.
OK! I learned something.
> e is a brother of f.
That's impossible!
```

Figure 15. Gender Restrictions in Asserting Rules

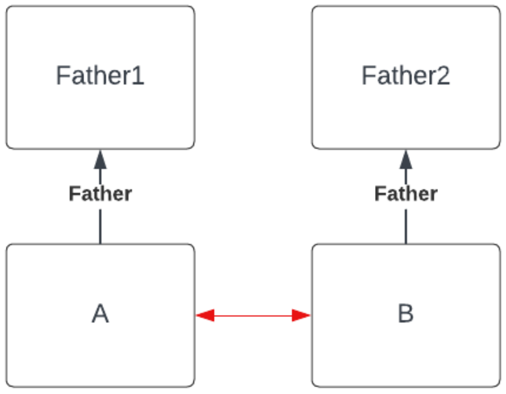
3. For the next test, we want to prevent the user from re-asserting rules, such as if “A” is already a parent of “B,” “A” can not be a sibling, pibling, or grandparent. **Figure 16** shows the validation for this test case.

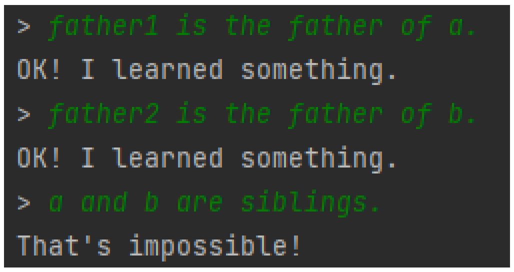


```
> a is the father of b.
OK! I learned something.
> a is an uncle of b.
That's impossible!
> a is a brother of b.
That's impossible!
> a and b are siblings.
That's impossible!
> a is a grandfather of b.
That's impossible!
```

Figure 16. Restricting Multiple Roles

4. In the next test cases, we wanted to prevent a child from having more than one father or more than 2 parents. In **Figure 17** are two people, a and b. Each has a different father. With this, a and b can not be siblings.

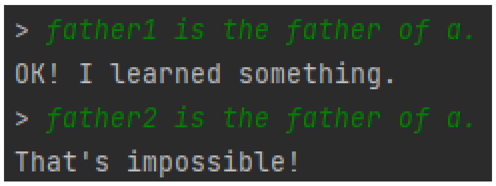




```
> father1 is the father of a.
OK! I learned something.
> father2 is the father of b.
OK! I learned something.
> a and b are siblings.
That's impossible!
```

Figure 17. Siblings Constraint if Both Children have similar parental roles

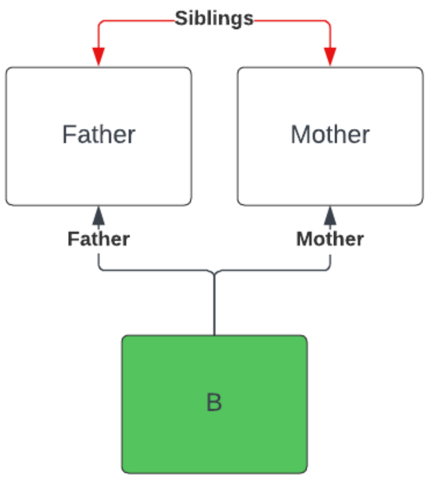
5. In **Figure 18**, we also tested if "A" has a father named father1, then it is impossible for "A" to have another father, which verifies this test case.

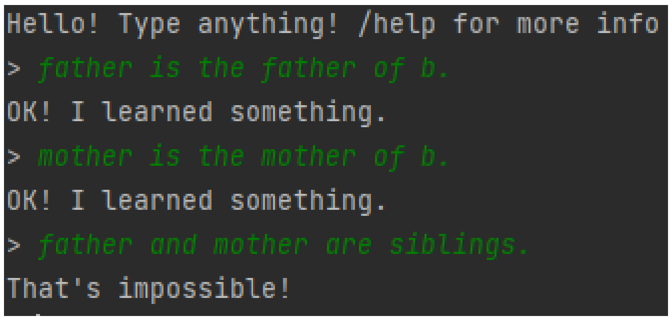


```
> father1 is the father of a.
OK! I learned something.
> father2 is the father of a.
That's impossible!
```

Figure 18. Single Parental Role Constraint

6. For the next test case, we wanted to create an error handling for couples, as we assumed that couples could not be siblings. In **Figure 19**, we were able to verify that this feature works.

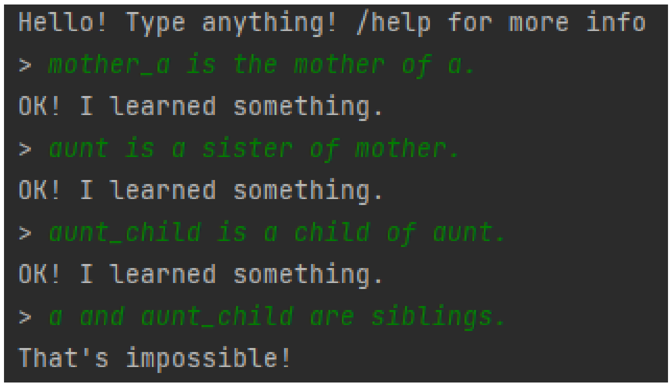




```
Hello! Type anything! /help for more info
> father is the father of b.
OK! I learned something.
> mother is the mother of b.
OK! I learned something.
> father and mother are siblings.
That's impossible!
```

Figure 19. Preventing Parents of Same Child to become Siblings

7. For the last error handling, we want to prevent sibling relationships with cousins. In Figure 20, we are able to establish this constraint.



```
Hello! Type anything! /help for more info
> mother_a is the mother of a.
OK! I learned something.
> aunt is a sister of mother.
OK! I learned something.
> aunt_child is a child of aunt.
OK! I learned something.
> a and aunt_child are siblings.
That's impossible!
```

Figure 20. Preventing Cousins from being Siblings

VI. Limitations and Challenges

The limitations the group set for the project are the "levels" of relationships. The available relationships are until grandparents only, meaning great-grandparents and ancestors are out of the scope of the group's implementation of the project. The project's scope is grandparents, parents, uncles and aunts, siblings, and cousins. When compared to large language models, the group's chatbot is only limited to familiar relationships, while large language models have access to knowledge bases that encompass most of the data on the internet, which is extremely vast.

The main challenge faced by the group is implementing the classifications of the inputted names by the user. For example, the initial creation of the project did not have a classification for cousins because it would need to check the parents of the two children and see if they are related. Another challenge faced was the limit of the number of parents, specifically given that a child can have a maximum number of 1 mother and father. The initial implementation of the project allowed multiple fathers and mothers for each child, making it confusing and unrealistic.

Another main challenge was creating constraints. Given the multiple possibilities of asserting a relationship, creating a constraint or error handling is vast. If the rules are too strict, there could be some valid assertions that could be invalidated, while if the rules are too loose, the rule might not cover all possibilities. Given this, multiple test cases have to be made, and for each possible constraint, a strict error handling specific to the test case has to be designed. And it is still possible that we were not able to cover a constraint or we made a constraint that covers something that shouldn't.

VII. Conclusion

In conclusion, the implementation of the chatbot for creating a family tree knowledge base was a simple but eye-opening experience into what chatbots are capable of in terms of handling complex queries about familial relationships. Something as trivial as a family and only as far back as the 3rd generation would result in a multitude of relationships, rules, and even assumptions. This shows both the capabilities of machine learning through a knowledge base and the complexity that is present in the real world. The project was implemented using Prolog and Python, and Depth First Search was used to drive knowledge from the knowledge base. This allowed the chatbot to navigate through connections even if not explicitly stated. The chatbot was also able to determine the gender of the individuals and can handle cases where it should be impossible for a particular relationship to exist

VIII. Contributions

Antonio Louix Ceriola	Percival Jr. Macalintal	Arjay Kyle Rabe
● ifRelated() method ● ifCousin() method ● Multiple children to one parent statement. ● Documentation ● Run test cases for program	● Refactoring code ● Fixing prompts ● Debugging ● Other helper methods	● Error handling for statement inputs ● Formulated conditions that prevent unrealistic data ● Fixing failed test cases ● Debugging

Jerald Krister Tabuzo	Ian Matthew Yanga
● findSiblings() method, dfs_siblings() method ● getSiblings() method ● Formulated conditions that prevent assertion of more than two parents, and more than 1 parental role. ● Lucidchart Diagram Representation of family tree in report.	● The Complete Test Cases ● Bug Finding ● Rechecking and Confirmation of the Validation of Functions/Relationships ● Proofreading and Reformatting

IX. References

Hingrajia, M. (2023). How do Chatbots Work? A Guide to Chatbot Architecture. Accessed on https://marutitech.com/chatbots-work-guide-chatbot-architecture/

Stackoverflow (2023). Assertion Failed when using Pyswip. Accessed on https://stackoverflow.com/questions/75785959/assertion-failed-when-using-pyswip

Yuce (2023). Pyswip Documentation. Accessed on https://github.com/yuce/pyswip