CSE 404

Artificial Intelligence Lab

God Hierarchies:
A Rule-Based Mythological Knowledge System

Submitted By:

Afrin Sultana Akhi (22101095)

Section**: B-2**

Semester**: 3-2**

Session**: Fall 2024**

Submitted To:

Lecturer

Nahida Marzan

Department of Computer Science and Engineering

University of Asia Pacific

Table of Contents

Problem Title

God Hierarchies: A Rule-Based Mythological Knowledge System in Prolog

This project builds a structured, queryable knowledge base representing mythological deities, their relationships, domains, and religious traditions using logical facts and inference rules in Prolog. The system enables recursive exploration of divine ancestry, domain-based deity lookup, and cross-religion analysis, illustrating how logic programming can model complex belief systems across cultures.

Problem Description

Throughout human history, mythology has played a significant role in shaping cultures, beliefs, and philosophies. Deities in various mythologies exhibit hierarchical relationships—such as parenthood, siblinghood, and succession—and embody specific domains such as war, wisdom, love, and the underworld.

However, comparing and analyzing such divine hierarchies across mythologies like Greek, Norse, Hindu, Egyptian, Celtic, Japanese, and Islamic traditions is often challenging due to the scattered nature of the data and symbolic variations.

God Hierarchies is a Prolog-based expert system designed to model and simulate deities from multiple world mythologies. It captures both genealogical (family-based) and functional (domain-based) relationships among gods, and allows recursive queries to reason about ancestry, domain overlap, and lineage across pantheons.

The system includes structured knowledge such as:

- Gods and their domains (e.g., thunder, war, fertility)
- Parent-child and sibling relationships
- Mythology or religion affiliation
- Ancestor/descendant relationships
- Shared domains across different mythologies

In addition to static facts, the system supports dynamic logical reasoning using rules such as:

- **ancestor**: recursively determine if one god is an ancestor of another
- **domain_overlap**: identify gods sharing similar domains

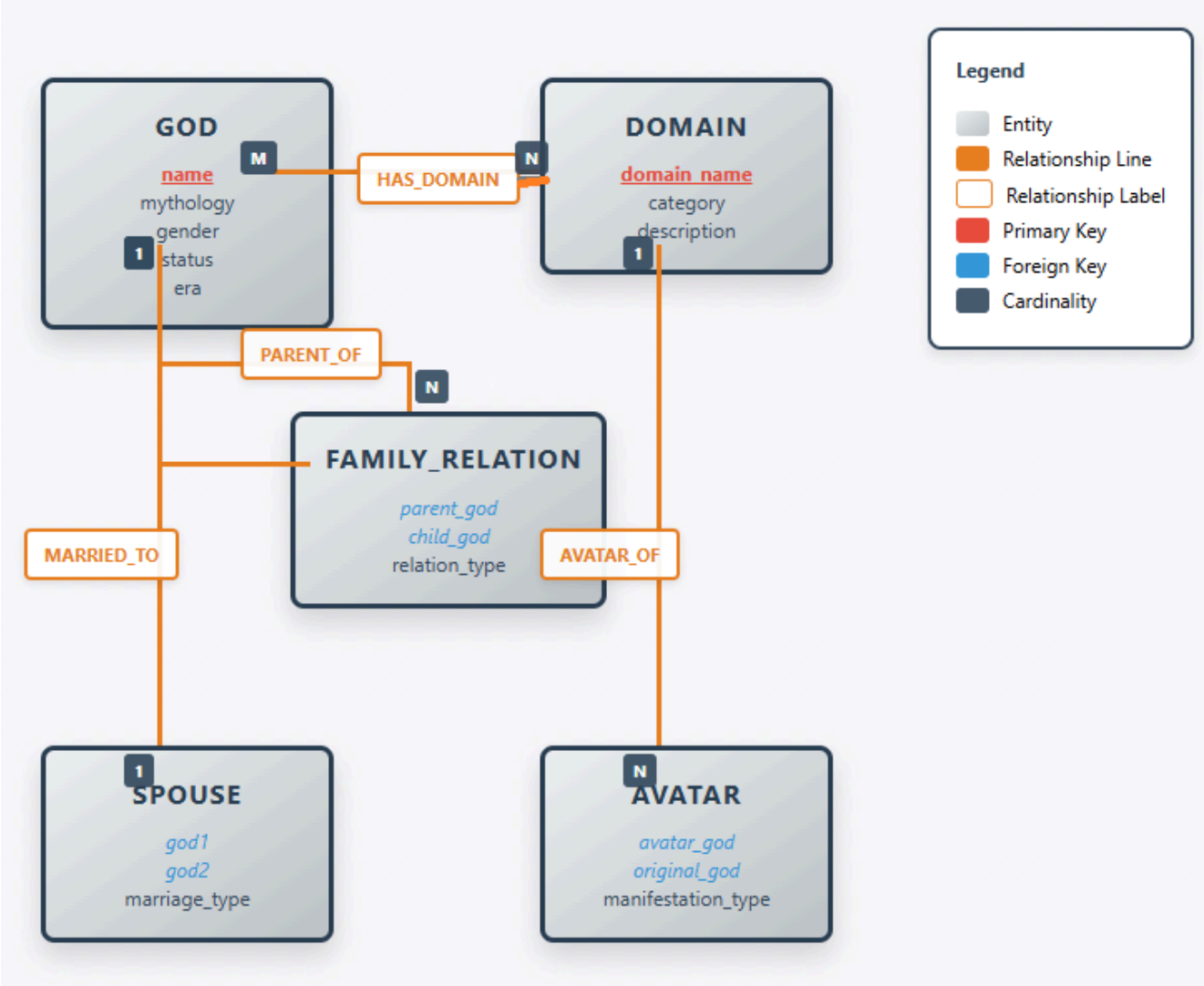
- **god_lineage**: display family hierarchies
- **belongs_to**: associate gods with a mythology/religion

This project demonstrates how symbolic logic in AI can represent complex, cross-cultural mythological systems and allow structured comparative analysis through Prolog.

Tools and Languages Used

- **Programming Language**: Prolog (SWI-Prolog)
- **Platform/IDE**: Visual Studio Code with SWI-Prolog Extension
- **Operating System**: Windows 11
- **Other Tools**: Online mythology references for data validation (e.g., Britannica, Wikipedia, Godchecker) , draw.io (for diagrams)

Diagram/Figure



Entity Descriptions:

- **GOD:** Core entity representing deities from various mythologies
- **DOMAIN:** Areas of power/influence (war, wisdom, love, etc.)
- **FAMILY_RELATION:** Parent-child relationships between gods
- **SPOUSE:** Marriage relationships between deities
- **AVATAR:** Divine manifestations (e.g., Rama as avatar of Vishnu)

Key Relationships:

- **HAS_DOMAIN (M:N):** Gods can have multiple domains, domains can belong to multiple gods
- **PARENT_OF (1:N):** One god can have many children
- **MARRIED_TO (1:1):** Monogamous marriage relationships
- **AVATAR_OF (N:1):** Multiple avatars can belong to one original deity

Sample Input/output.]

RECURSIVE RULES :

1. **descendants(A, Descendants) :- findall(D, descendant(D, A), Descendants).**
   - Purpose: Collects all descendants of a deity. (Recursive)
   - Query: descendants(zeus, List).

```
?-  descendants(zeus, List).
List = [athena, apollo, artemis, ares, hephaestus, dionysus, persephone].
```

2. **count_descendants(A, Count) :- descendants(A, Descendants), length(Descendants, Count).**
   - Purpose: Counts all descendants of a deity. (Recursive)
   - Query: count_descendants(chronus, N), write('Descendants of Chronus: '), write(N), nl

```
?- count_descendants(chronus, N), write('Descendants of Chronus: '), write(N), nl.
Descendants of Chronus: 16
N = 16.
```

3. **lineage(D, [D|Rest]) :- parent(P, D), lineage(P, Rest).**
   lineage(D, [D]) :- \+ has_parents(D).
   - Purpose: Builds a list of ancestors up to the root. (Recursive)
   - Query: lineage(apollo, Line), write('Lineage of Apollo: '), write(Line), nl.

Lineage of Apollo: [apollo,zeus,chronus]

```
?- lineage(apollo, Line), write('Lineage of Apollo: '), write(Line), nl.
Lineage of Apollo: [apollo,zeus,chronus]
Line = [apollo, zeus, chronus] .
```

4. **child_count(Parent, Count) :- findall(Child, parent(Parent, Child), Children), length(Children, Count)**.
   - Purpose: Counts the number of children a deity has.
   - Query: child_count(zeus, N).

```
?- child_count(zeus, N).
N = 7.
```

5. **orphan(X) :- god(X, _, _, _, _), \+ has_parents(X).**
   - Purpose: True if X has no known parents.
   - Query: orphan(athena).

```
?- orphan(athena)
false.
```

6. **moon_deity(X) :- domain(X, moon).**
   - Purpose: True if X is a moon deity.
   - Query: moon_deity(tsukuyomi).

```
?- moon_deity(tsukuyomi).
true.
```

CLASSIFICATION RULES:

7. **same_mythology(X, Y) :- god(X, M, _, _, _), god(Y, M, _, _, _), X \= Y.**
   - Purpose: True if X and Y belong to the same mythology.
   - Query: same_mythology(zeus, hera).

```
?- same_mythology(zeus, hera).
true.
```

8. **japanese_god(X) :- god(X, japanese, _, _, _).**
   - Purpose: True if X is a Japanese god.
   - Query: japanese_god(amaterasu).

```
?- japanese_god(amaterasu)
true.
```

9. **ancestor(A, D) :- parent(A, D).**
   ancestor(A, D) :- parent(A, X), ancestor(X, D).
   - Purpose: True if A is any ancestor (parent, grandparent, etc.) of D. (Recursive)

- Query: ancestor(chronus, zeus).

```
?-  ancestor(chronus, zeus)
true
```

10. **parent(X, Y) :- father(Y, X).**
    parent(X, Y) :- mother(Y, X).
    - Purpose: True if X is a parent of Y (either father or mother).
    - Query: parent(zeus, ares)

```
?-  parent(zeus, ares)
|    .
true
```

12. **sibling(X, Y) :- parent(P, X), parent(P, Y), X \= Y.**
    - Purpose: True if X and Y share at least one parent.
    - Query: sibling(zeus, poseidon).

```
?-  sibling(zeus, poseidon)
true
```

```
?- forall(female_god(God), (write('Female: '), write(God), nl)).
Female: hera
Female: rhea
Female: demeter
Female: athena
Female: artemis
Female: aphrodite
Female: hestia
Female: persephone
Female: frigg
Female: freya
Female: saraswati
Female: lakshmi
Female: parvati
Female: amaterasu
Female: izanami
Female: isis
Female: bastet
Female: nephthys
Female: brigid
Female: morrigan
Female: danu
true.
```

```
?- findall(God, greek_god(God), List), length(List, N), write('Greek gods: '), write(N), nl.
Greek gods: 17
List = [zeus, hera, chronus, rhea, hades, poseidon, demeter, athena, apollo|...],
N = 17.
```

```
?- family_tree(zeus).

â▌▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌
â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌▌
â▌´              FAMILY TREE OF zeusâ▌´
â▌ â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌
â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌£
â▌´ Father: chronus                          â▌´
â▌´ Mother: rhea                             â▌´
â▌´ Spouse: hera                             â▌´
â▌´ Siblings: [hades,poseidon,hera,demeter,hestia,hades,poseidon,hera,demeter,hestia]â▌´
â▌´ Children: [athena,apollo,artemis,ares,hephaestus,dionysus,persephone]â▌´
â▌▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌
â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌â▌
true.
```

Static Data Queries

1. show_domain_gods(death).
2. forall(female_god(God), (write('Female: '), write(God), nl)).
3. family_tree(zeus)
4. forall((female_god(G), supreme_deity(G)), write(G)).
5. forall(domain(God, Domain), (write(God), write(' -> '), write(Domain), nl)).
6. list_gods_from(japanese).
7. forall(domain(God, Domain), (write(God), write(' -> '), write(Domain), nl)).
8. prophet(muhammad).
9. Count all supreme deities:
10. findall(God, supreme_deity(God), List), length(List, N), write('Supreme deities: '), write(N), nl.
11.

Conclusion

This project successfully shows how logic programming can be applied to humanities-based domains such as mythology. Using SWI-Prolog, we built a structured knowledge base that supports recursive queries, multi-religious comparisons, and complex relationship tracing.

From determining divine ancestry to finding gods who share similar domains across mythologies, this system demonstrates how AI techniques can preserve, study, and analyze cultural knowledge logically and interactively.

It also opens doors for educational tools, cross-cultural studies, and symbolic reasoning research in the field of digital humanities.

Challenges Faced

Recursive Relationship Handling

Implementing rules like `ancestor/2` and `god_lineage/1` required careful attention to recursion and base cases to prevent infinite loops and stack overflows.

Domain Redundancy

Since multiple gods across different religions can have overlapping domains (e.g., thunder, war), ensuring non-ambiguous reasoning required clean and normalized domain assignments.

Output Formatting in Prolog

Getting clean, readable output (like line-separated deity names instead of a single long string) involved adjusting list-processing logic and using `forall/2`

Cultural Sensitivity and Accuracy

Including figures from active religions (like Islamic figures) had to be handled carefully and respectfully, ensuring they were presented in an informational rather than theological manner.

Syntax Debugging

As with any Prolog project, forgetting to declare facts or misusing recursion led to frequent issues—resolved via modular testing and query tracing.

GitHub Link : https://github.com/SnoOpWoOt/God-Hierarchy-Prolog.