NeuroSymbolic Commonsense Reasoner

Project 1: Text-Based Reasoning Module

CS-F429 Course Project | November 25, 2025

Introduction & Core Objectives

Project Goal

To build a **NeuroSymbolic pipeline** capable of commonsense reasoning. The system is designed to bridge the gap between the flexibility of neural models and the rigor of symbolic logic.

Task 1 Approach

We focus exclusively on text-based reasoning using the **CLUTRR dataset**. Our strategy combines LLM-based semantic parsing (Mistral-7B) with deterministic symbolic inference (Prolog) to ensure accuracy and explainability.

Problem Statement

Convert natural-language family stories into structured predicates, perform multi-hop logical inference, and generate a final relationship with a Chain-of-Thought explanation.

The CLUTRR Dataset

id string · *lengths* 36 – 36	story string · *lengths* 45 – 769	query string · *lengths* 14 – 27	target int32 0 – 17	target_text string · *classes* 18 values
0fc660c1-e7d5-41fb-8d72-…	[Ashley]'s daughter,…	('Ashley', 'Nicholas')	15	son
b1c985cf-69b5-4575-97d3-…	[Nancy] likes to cut the hair of…	('Nancy', 'Lorraine')	16	daughter
5e7fc867-3782-4ff6-98d5-…	[Dale] and his sister [Nancy] ar…	('Dale', 'Louise')	17	niece
9bc318aa-3c85-4fef-8e6f-…	[Lillian] and her sister [Nancy] ar…	('Nancy', 'Douglas')	14	nephew
6cb675ce-a3d1-44ca-8d3e-…	[Ashley] liked to go to the park…	('Dale', 'Ashley')	6	mother
689c12a6-d83d-	[Theresa] wants to	('Ashley',	11	granddaughter

Compositional Language Understanding and Text-based Relational Reasoning

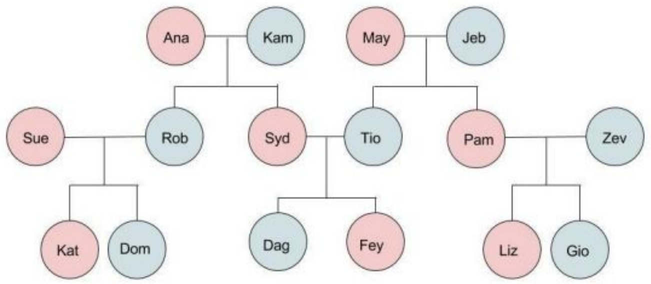
- **Content:** Short stories describing complex family relationships.
- **Complexity:** Requires 2 to 4 hops of reasoning to solve.
- **Objective:** Identify the specific relationship between two target entities within the narrative.
- **Role:** Serves as the ground truth for testing our neuro-symbolic extraction capabilities.

Architecture Overview

MISTRAL AI_

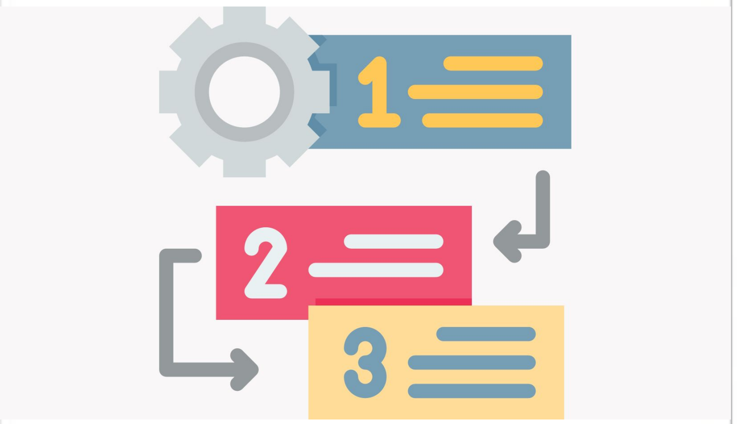
1. Semantic Extraction

Using Mistral-7B Instruct to parse natural language into structured logic predicates.



2. Symbolic Reasoning

Executing multi-hop inference using a local SWI-Prolog solver for deterministic results.



3. CoT Generation

Verbalizing the Prolog solver traces back into human-readable Chain-of-Thought explanations.

Stage 1: Semantic Extraction

Using Mistral-7B Instruct

We employ a frozen Mistral-7B model to extract logical predicates from the story text. To ensure compatibility with our solver, we enforced strict constraints:

Prompt Engineering Strategy:

- Closed vocabulary of relations (e.g., father, aunt).
- Python-style list output format.
- Two-pass COT pipeline: Summary → Predicates.

Query extraction:

- A simple multi-shot prompt with suitable examples was sufficient.

```
actual story :  [Lynn] loves cooking for her son. His name is [Thomas] [Thomas] wanted to go to his gr
andfather [James] 'house because he always has a good time when he is there. [James] bought a new dres
s for his daughter [Lena].
Running model1......................................................................

The following generation flags are not valid and may be ignored: ['temperature']. Set `TRANSFORMERS_VE
RBOSITY=info` for more details.

    1. Lynn is Thomas's mother.
    2. Thomas wants to visit James.
    3. James is Lena's father.
    4. Lena received a new dress from James.
Running model2.................................................................

The following generation flags are not valid and may be ignored: ['temperature']. Set `TRANSFORMERS_VE
RBOSITY=info` for more details.

    sections:
    ---------------------------------------------------------
    Genders:
    lynn=female.
    thomas=male.
    james=male.
    lena=female.
    ---------------------------------------------------------
    Triplets:
    (thomas, mother, lynn),
    (james, father, lena)
    ---------------------------------------------------------
    Facts:
    father(james, lena).
    mother(lynn, thomas).
    male(thomas).
    female(lynn).
    male(james).
    female(lena).
```

Extraction Challenges & Solutions

Hallucinations & Formatting: LLMs often produce verbose text or hallucinated relationships outside the schema.
Solution: Extract valid subject-object pairs and filter against ALLOWED_INPUTS.

Missing Predicates: If a relationship type (e.g., "uncle") isn't in the story, the Prolog solver crashes on undefined predicates.
Solution: Automatic dummy fact injection (e.g., uncle(dummy1, dummy2)) ensures the Knowledge Base is complete.

Syntax Drift: Ensuring the output is strictly valid Prolog code.
Solution: Strict few-shot prompting with explicit negative constraints (e.g., "No conversational text") and a two step pipeline.

LLM Stability: Trying to maintain only the required context in the LLM.
Solution: Two step pipeline (Story -> Summary -> Predicates)

Stage-1.1 Prompts

```
p1 = """You are a family-relationship extraction engine.

Your job is to read the story and rewrite it into relationship bullet points that contains:
- only and all family relationships
- gender every new person you come across using context - dont base it off the name
- NO events, objects, activities, locations, or irrelevant descriptions
- NO timeline or sequencing details
- NO pronouns; always replace pronouns with the correct resolved person name
- NO hallucinations - include ONLY what is explicitly stated or directly implied
- If a relationship is ambiguous, OMIT it.


extract relation phrases from the story after excluding the unnecesary details using all previous context :

{story}
FINISH
"""
```

Stage-1.1

```
actual story :  [Emma] went to the garden to find her father [John]. [John] was talking to his brother
[Peter]. [Peter] mentioned that their mother [Sarah] was inside watching TV. [Emma]'s sister [Kate] ra
n past them playing tag. Later, [John]'s wife [Lisa] called everyone for lunch.

Running model1.....................................................................

The following generation flags are not valid and may be ignored: ['temperature']. Set `TRANSFORMERS_VE
RBOSITY=info` for more details.

    1. Emma is the daughter of John.
    2. John is the father of Emma and the brother of Peter.
    3. Peter is the brother of John and the father of Kate.
    4. Sarah is the mother of John and Peter.
    5. Kate is the sister of Emma and the daughter of Peter.
    6. Lisa is the wife of John.
```

Stage-1.2 Prompts

```
p2 = """You are a family-relationship mapper.
Your task is to read the cleaned family summary and output ONLY three sections:
Genders, Triplets, Facts.

You MUST think step-by-step internally, but DO NOT reveal your reasoning.
Output ONLY the three sections exactly as described.

=========================================================
RULES
=========================================================
1. Names refer to unique people.
2. Use only explicit relationships from the summary.
3. Infer gender from roles (father, mother, son, daughter, husband, wife, brother, sister) DO NOT INFER FROM THEIR NAMES!.
4. DO NOT guess information that is not stated.
5. DO NOT create placeholder names or "unknown".
6. Use the most specific relation available.
7. In the Facts section, include gender as:
   male(name).  female(name).  unknown_gender(name).
   (Use lowercase names as atoms.)

=========================================================
OUTPUT SPECIFICATION
=========================================================

### SECTION 1 – Genders
Format: name=male/female/unknown

### SECTION 2 – Triplets
Format: (subject, relation, object), relations are only family relations, ignore others:
ONLY Allowed relation labels:
('granddaughter', 'father', 'brother', 'mother', 'grandson', 'sister', 'uncle', 'daughter', 'husband', 'grandmother', 'wife', 'aunt', 'son', 'grandfather')

### SECTION 3 – Facts
Convert both gender and triplets into Prolog facts.

For gender:
- male(name).
- female(name).
- unknown_gender(name).

For relations:
father(subject, object),
brother(subject, object), etc

Use lowercase atoms:
Example: father(don, steve).

=========================================================
NOW PROCESS THESE RELATIONS into sections:
{story}
RELATIONS FINISH
sections:
"""
```

Stage-1.2

```
Running model2.................................................................

    sections:
    --------------------------------------------------------
    Genders:
    Emma=female,
    John=male,
    Peter=male,
    Sarah=female,
    Kate=female,
    Lisa=unknown

    Triplets:
    (John, father, Emma),
    (John, brother, Peter),
    (Peter, father, Kate),
    (Peter, brother, John),
    (Sarah, mother, John),
    (Sarah, mother, Peter),
    (Kate, sister, Emma),
    (Kate, daughter, Peter)

    Facts:
    male(john).
    female(emma).
    female(sarah).
    male(peter).
    female(kate).
    unknown_gender(lisa).
    father(john, emma).
    brother(john, peter).
    father(peter, kate).
    brother(peter, john).
    mother(sarah, john).
    mother(sarah, peter).
    sister(kate, emma).
    daughter(kate, peter)
```

Stage-1.3

```python
def to_prolog_query(question: str):
    prompt = f"""
Convert the following natural-language question into a Prolog query.
Use ONLY standard Prolog syntax.

RULES:
- Use lowercase atoms for predicates and constants.
- Replace spaces in names with underscores.
- For yes/no questions, generate a predicate that logically matches the question.
- Output ONLY the Prolog query, nothing else.

EXAMPLES:
EXAMPLES:

Q: "Is Alice Bob's mother?"
A: mother(alice, bob).

Q: "Is Bob Alice's father?"
A: father(bob, alice).

Q: "Is John Mary's brother?"
A: brother(john, mary).

Q: "Is Sarah Emily's sister?"
A: sister(sarah, emily).

Q: "Is Claire the grandmother of David?"
A: grandmother(claire, david).

Q: "Is Helen David's granddaughter?"
A: granddaughter(helen, david).

Q: "Is Michael Jane's son?"
A: son(michael, jane).

Q: "Is Jane Michael's daughter?"
A: daughter(jane, michael).



NOW CONVERT:
Q: "{question}"
A:
"""
    return generate(prompt, max_new_tokens=50, temperature=0.001)
```

```python
query = to_prolog_query("is helena chloe's mother in law")
print(query.split("A:")[-1])
```

```
The following generation flags are not valid and may be ignored: ['temperature']. Set `TRANSFORMERS_VE
RBOSITY=info` for more details.


mother_in_law(helena, chloe).
```

Stage 2: Symbolic Reasoning

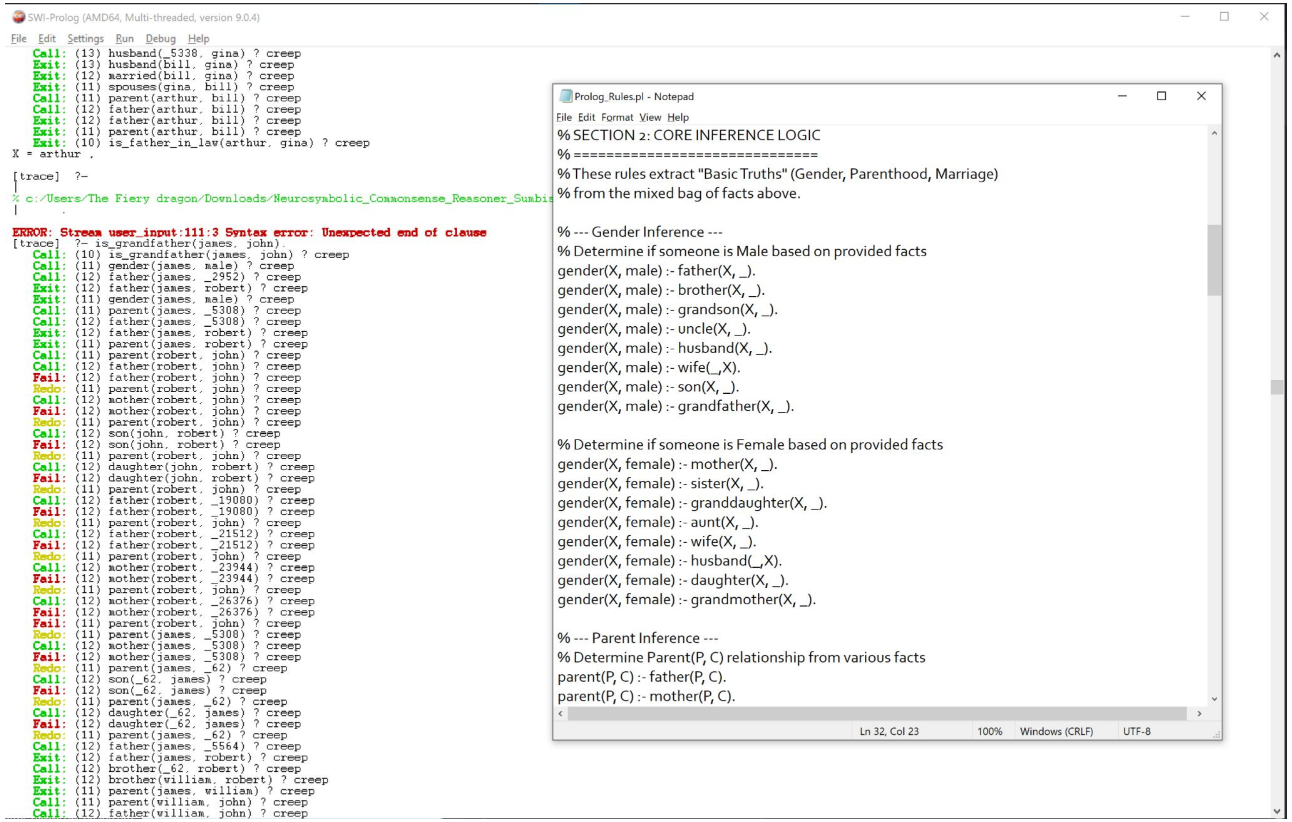
We chose **Prolog** over Z3 or MiniZinc for three key reasons:

1. **Trace Generation:** Prolog natively supports SLD resolution, providing a clear proof trace essential for CoT.
2. **Logic Fit:** Perfectly handles the multi-hop recursive logic required for family trees.

Note: None of Z3, MiniZinc, PySwip, PyDatalog, SWI-Prolog were both lightweight enough to run on notebooks and had native trace support (Support was removed for most versions and we faced dependency issues while downgrading)

The sanitized predicates are written to a .pl file alongside a static ruleset (rules.pl).

SWI Prolog



Inference Logic

- father
- mother
- brother
- sister
- grandson
- granddaughter
- uncle
- aunt
- husband
- wife
- grandmother
- grandfather
- son
- daughter
- mother-in-law
- father-in-law

Raw Input Facts	→	**The 3 Core Truths (Normalized)**
`father`, `son`, `brother` ...	→	`gender(Person, Type)` *(Is this person male or female?)*
`mother`, `daughter`, `father` ...	→	`parent(Parent, Child)` *(Who spawned whom?)*
`husband`, `wife`	→	`spouses(PersonA, PersonB)` *(Who is married?)*

Inference Logic

A. The Simple Combinations (Direct Lookup)

These rules simply apply a "Gender Filter" to a "Relationship Link".

- `is_father(X, Y)`

$$= \text{gender}(X, \text{male}) + \text{parent}(X, Y)$$

(Logic: A male who is a parent.)

```
% 1. FATHER: A male parent
is_father(X, Y) :-
  gender(X, male),
  parent(X, Y).
```

B. The "Triangulated" Combinations (Siblings)

These rules are smarter. Since you don't have a "sibling" core truth, you construct it by finding a **shared parent**.

- `is_sister(X, Y)`

$$= \text{gender}(X, \text{female}) + \text{parent}(Z, X) + \text{parent}(Z, Y)$$

```
% 8. SISTER: Female sibling (shares at least one parent)
is_sister(X, Y) :-
  gender(X, female),
  parent(Z, X),
  parent(Z, Y),
  X \= Y.
```

Inference Logic

C. The "Chained" Combinations (Grandparents)

These rules work by "hopping" over an intermediate person.

- `is_grandfather(X, Y)`

$$= \text{gender}(X, \text{male}) + \text{parent}(X, Z) + \text{parent}(Z, Y)$$

(Logic: A male who is the parent of someone (Z), who is the parent of Y.)

```
% 9. GRANDFATHER: Male parent of a parent
is_grandfather(X, Y) :-
  gender(X, male),
  parent(X, Z),
  parent(Z, Y).

is_grandfather(X, Z) :-
  gender(X, male),
  grandfather(X, Y),
  brother(Y, Z).

is_grandfather(X, Z) :-
  gender(X, male),
  grandfather(X, Y),
  sister(Y, Z).
```

D. The "Hybrid" Combinations (In-Laws)

This is your most complex rule type. It combines a **Spouse Link** with a **Parent Link**.

- `is_mother_in_law(X, Y)`

$$= \text{gender}(X, \text{female}) + \text{spouses}(Y, Z) + \text{parent}(X, Z)$$

(Logic: A female who is the parent of the person (Z) that Y is married to.)

```
% 16. MOTHER-IN-LAW: Mother of one's spouse
is_mother_in_law(X, Y) :-
  gender(X, female),   % X must be female
  spouses(Y, Z),       % Y is married to Z
  parent(X, Z).        % X is the parent of Z (the spouse)
```

```
[trace]  ?- is_grandfather(james, john).
   Call: (10) is_grandfather(james, john) ? creep
   Call: (11) gender(james, male) ? creep
   Call: (12) father(james, _2952) ? creep
   Exit: (12) father(james, robert) ? creep
   Exit: (11) gender(james, male) ? creep
   Call: (11) parent(james, _5308) ? creep
   Call: (12) father(james, _5308) ? creep
   Exit: (12) father(james, robert) ? creep
   Exit: (11) parent(james, robert) ? creep
   Call: (11) parent(robert, john) ? creep
   Call: (12) father(robert, john) ? creep
   Fail: (12) father(robert, john) ? creep
   Redo: (11) parent(robert, john) ? creep
   Call: (12) mother(robert, john) ? creep
   Fail: (12) mother(robert, john) ? creep
   Redo: (11) parent(robert, john) ? creep
   Call: (12) son(john, robert) ? creep
   Fail: (12) son(john, robert) ? creep
   Redo: (11) parent(robert, john) ? creep
   Call: (12) daughter(john, robert) ? creep
   Fail: (12) daughter(john, robert) ? creep
   Redo: (11) parent(robert, john) ? creep
   Call: (12) father(robert, _19080) ? creep
   Fail: (12) father(robert, _19080) ? creep
   Redo: (11) parent(robert, john) ? creep
   Call: (12) father(robert, _21512) ? creep
   Fail: (12) father(robert, _21512) ? creep
   Redo: (11) parent(robert, john) ? creep
   Call: (12) mother(robert, _23944) ? creep
   Fail: (12) mother(robert, _23944) ? creep
   Redo: (11) parent(robert, john) ? creep
   Call: (12) mother(robert, _26376) ? creep
   Fail: (12) mother(robert, _26376) ? creep
   Fail: (11) parent(robert, john) ? creep
   Redo: (11) parent(james, _5308) ? creep
   Call: (12) mother(james, _5308) ? creep
   Fail: (12) mother(james, _5308) ? creep
   Redo: (11) parent(james, _62) ? creep
   Call: (12) son(_62, james) ? creep
   Fail: (12) son(_62, james) ? creep
   Redo: (11) parent(james, _62) ? creep
   Call: (12) daughter(_62, james) ? creep
   Fail: (12) daughter(_62, james) ? creep
   Redo: (11) parent(james, _62) ? creep
   Call: (12) father(james, _5564) ? creep
   Exit: (12) father(james, robert) ? creep
   Call: (12) brother(_62, robert) ? creep
   Exit: (12) brother(william, robert) ? creep
   Exit: (11) parent(james, william) ? creep
   Call: (11) parent(william, john) ? creep
   Call: (12) father(william, john) ? creep
   Fail: (12) father(william, john) ? creep
   Redo: (11) parent(william, john) ? creep
   Call: (12) mother(william, john) ? creep
   Fail: (12) mother(william, john) ? creep
   Redo: (11) parent(william, john) ? creep
   Call: (12) son(john, william) ? creep
   Fail: (12) son(john, william) ? creep
   Redo: (11) parent(william, john) ? creep
   Call: (12) daughter(john, william) ? creep
   Fail: (12) daughter(john, william) ? creep
   Redo: (11) parent(william, john) ? creep
   Call: (12) father(william, _20146) ? creep
   Fail: (12) father(william, _20146) ? creep
   Redo: (11) parent(william, john) ? creep
```

```
   Call: (12) son(john, william) ? creep
   Fail: (12) son(john, william) ? creep
   Redo: (11) parent(william, john) ? creep
   Call: (12) daughter(john, william) ? creep
   Fail: (12) daughter(john, william) ? creep
   Redo: (11) parent(william, john) ? creep
   Call: (12) father(william, _20146) ? creep
   Fail: (12) father(william, _20146) ? creep
   Redo: (11) parent(william, john) ? creep
   Call: (12) father(william, _22578) ? creep
   Fail: (12) father(william, _22578) ? creep
   Redo: (11) parent(william, john) ? creep
   Call: (12) mother(william, _25010) ? creep
   Fail: (12) mother(william, _25010) ? creep
   Redo: (11) parent(william, john) ? creep
   Call: (12) mother(william, _27442) ? creep
   Fail: (12) mother(william, _27442) ? creep
   Fail: (11) parent(william, john) ? creep
   Redo: (11) parent(james, _62) ? creep
   Call: (12) father(james, _30684) ? creep
   Exit: (12) father(james, robert) ? creep
   Call: (12) sister(_62, robert) ? creep
   Exit: (12) sister(mary, robert) ? creep
   Exit: (11) parent(james, mary) ? creep
   Call: (11) parent(mary, john) ? creep
   Call: (12) father(mary, john) ? creep
   Fail: (12) father(mary, john) ? creep
   Redo: (11) parent(mary, john) ? creep
   Call: (12) mother(mary, john) ? creep
   Fail: (12) mother(mary, john) ? creep
   Redo: (11) parent(mary, john) ? creep
   Call: (12) son(john, mary) ? creep
   Exit: (12) son(john, mary) ? creep
   Exit: (11) parent(mary, john) ? creep
   Exit: (10) is_grandfather(james, john) ? creep
true .
```

```
Exit: (12) father(james, robert) ? creep
Exit: (11) gender(james, male) ? creep
Exit: (12) father(james, robert) ? creep
Exit: (11) parent(james, robert) ? creep
Exit: (12) father(james, robert) ? creep
Exit: (12) brother(william, robert) ? creep
Exit: (11) parent(james, william) ? creep
Exit: (12) father(james, robert) ? creep
Exit: (12) sister(mary, robert) ? creep
Exit: (11) parent(james, mary) ? creep
Exit: (12) son(john, mary) ? creep
Exit: (11) parent(mary, john) ? creep
Exit: (10) is_grandfather(james, john) ? creep
```

Stage 3: Chain-of-Thought (CoT)

Prolog Trace

The solver outputs a raw logical trace of the derivation steps

Verbalization

Mistral-7B takes the trace and translates the logical steps into natural language sentences with COT.

Final Output

A human-friendly explanation (CoT) accompanying the final relationship answer.

Stage-3 Prompts

```
def summarize_prolog_trace_cot(trace: str):
    prompt = f"""
Summarize the following exit-only Prolog trace. First, think step-by-step
about what each exit line implies logically. Then, after reasoning, produce
a clean final paragraph that explains the reasoning chain in natural language.

RULES:
- Use lowercase atoms as they appear.
- DO NOT output the reasoning steps in the final answer.
- Output ONLY the final summarized paragraph.
- Your reasoning should interpret each exit line in order.

EXAMPLES:

TRACE:
Exit: mother(alice, beth)
Exit: parent(beth, claire)
Exit: grandmother(alice, claire)

THINK:
- mother(alice, beth) means Alice is Beth's mother.
- parent(beth, claire) means Beth is Claire's parent.
- Combining them shows Alice is the grandmother of Claire through Beth.

SUMMARY:
Alice is Claire's grandmother because she is the mother of Beth, and Beth is Claire's parent, forming a d
irect maternal lineage.

---

TRACE:
Exit: father(john, mark)
Exit: parent(mark, olivia)
Exit: grandfather(john, olivia)

THINK:
- father(john, mark) → John is Mark's father.
- parent(mark, olivia) → Mark is Olivia's parent.
- Combine → John is Olivia's grandfather through Mark.

SUMMARY:
John is Olivia's grandfather since he is Mark's father, and Mark is Olivia's parent, creating a paternal
line connecting John and Olivia.
```

```
TRACE:
Exit: sibling(sarah, michael)
Exit: parent(michael, emma)
Exit: aunt(sarah, emma)

THINK:
- Sarah and Michael are siblings.
- Michael is Emma's parent.
- A sibling of a parent is an aunt.

SUMMARY:
Sarah is Emma's aunt because she is Michael's sibling, and Michael is Emma's parent.

---

TRACE:
Exit: parent(linda, jack)
Exit: sibling(linda, robert)
Exit: parent(robert, claire)
Exit: cousin(jack, claire)

THINK:
- Linda is Jack's parent.
- Linda and Robert are siblings.
- Robert is Claire's parent.
- Children of siblings are cousins.

SUMMARY:
Jack and Claire are cousins because their parents, Linda and Robert, are siblings.

---

NOW SOLVE:
TRACE:
{trace}

THINK:
"""
    return generate(prompt, max_new_tokens=300, temperature=0.001)
```

Stage-3

In [12]:
```
import re

def extract_exit_traces(trace: str) -> str:
    exit_lines = re.findall(r"^\s*Exit:.*", trace, flags=re.MULTILINE)
    new_traces = "\n".join(exit_lines)
    return new_traces

new_traces = extract_exit_traces(trace)
print(new_traces)
```

```
   Exit: (12) mother(sarah, john) ? creep
   Exit: (11) gender(sarah, female) ? creep
   Exit: (12) mother(sarah, john) ? creep
   Exit: (11) parent(sarah, john) ? creep
   Exit: (12) father(john, emma) ? creep
   Exit: (11) parent(john, emma) ? creep
   Exit: (10) is_grandmother(sarah, emma) ? creep
```

In [13]:
```
COT = summarize_prolog_trace_cot(new_traces)
```

```
The following generation flags are not valid and may be ignored: ['temperature']. Set `TRANSFORMERS_VE
RBOSITY=info` for more details.
```

In [14]:
```
print(COT.split("THINK:")[-1])
```

```
- Sarah is John's mother.
- Sarah is female.
- Sarah is John's parent.
- John is Emma's father.
- John is Emma's parent.
- Combining them shows Sarah is Emma's grandmother.

SUMMARY:
Sarah is Emma's grandmother because she is John's mother, and John is Emma's parent, forming a direct
maternal lineage.
```

Thank you!
Questions?