Knowledge Representation

Introduction :

Knowledge representation in artificial intelligence (AI) is the process of encoding information into a format that AI can use to make decisions. It's a key component of AI, allowing systems to interpret data, draw conclusions, and solve problems.

Purpose of Each File:

Logic.py :

The core library defines the **'Sentence**' class and its derived subclasses, which are used to represent logical knowledge in object form. To determine logical entailment, the '**model_check'** function systematically evaluates all potential truth assignments.

How It Operates:

- Logical sentences are constructed using the Sentence class and its associated subclasses. These sentences can be combined using logical operators.
- The model_check function is applied to assess whether a query logically follows from a given knowledge base by testing all possible truth assignments.
- This method enables the program to infer new facts, solve logical problems, and validate or refute logical statements effectively.

clue.py :

The program tackles a version of the Clue game by determining the suspect, weapon, and room involved in the crime. It represents the relationships and constraints between these elements as logical sentences within the knowledge base (KB).

How It Works:

- Symbols such as *ColMustard* (suspect), *kitchen* (room), and *knife* (weapon) are used to represent the entities.
- Rules are added to ensure only one suspect, one room, and one weapon can be the solution.
- Known facts, like "*ColMustard is not the suspect,*" are included in the KB.
- Each symbol is analyzed to conclude whether it is definitely true, definitely false, or still uncertain based on the provided and derived information.

```python
import termcolor

from logic import *

mustard = Symbol("ColMustard")
plum = Symbol("ProfPlum")
scarlet = Symbol("MsScarlet")
characters = [mustard, plum, scarlet]

ballroom = Symbol("ballroom")
kitchen = Symbol("kitchen")
library = Symbol("library")
rooms = [ballroom, kitchen, library]

knife = Symbol("knife")
revolver = Symbol("revolver")
wrench = Symbol("wrench")
weapons = [knife, revolver, wrench]

symbols = characters + rooms + weapons


def check_knowledge(knowledge):
    for symbol in symbols:
        if model_check(knowledge, symbol):
            termcolor.cprint(f"{symbol}: YES", "green")
        elif not model_check(knowledge, Not(symbol)):
            print(f"{symbol}: MAYBE")


# There must be a person, room, and weapon.
knowledge = And(
    Or(mustard, plum, scarlet),
    Or(ballroom, kitchen, library),
    Or(knife, revolver, wrench)
)
```

```python
    # Initial cards
    knowledge.add(And(
        Not(mustard), Not(kitchen), Not(revolver)
    ))

    # Unknown card
    knowledge.add(Or(
        Not(scarlet), Not(library), Not(wrench)
    ))

    # Known cards
    knowledge.add(Not(plum))
    knowledge.add(Not(ballroom))

    check_knowledge(knowledge)
```

Result:

```
PS C:\Users\mmd14\OneDrive\Desktop\knowledge Representation>
src/clue.py
MsScarlet: YES
library: YES
knife: YES
PS C:\Users\mmd14\OneDrive\Desktop\knowledge Representation>
```

harry.py:

The program solves a simple logic puzzle using characters from Harry Potter. It uses logic to determine if it's raining based on rules and facts.

How It Works:

- Adds rules like: “If it doesn’t rain, Hagrid is present.”
- Includes facts like: “Dumbledore is present.”
- Tests whether these facts logically imply that it must be raining.

```python
from logic import *

rain = Symbol("rain")
hagrid = Symbol("hagrid")
dumbledore = Symbol("dumbledore")

knowledge = And(
    Implication(Not(rain), hagrid),
    Or(hagrid, dumbledore),
    Not(And(hagrid, dumbledore)),
    dumbledore
)

print(model_check(knowledge, rain))
```

Result :

```
PS C:\Users\mmd14\OneDrive\Desktop\knowledge Representation> 
entation/src/harry.py"
True
```

mastermind.py : The program solves a simplified Mastermind game by using logic to figure out the correct color combination.

How It Works:

- Represents color-position pairs with symbols like *red0* (red is in position 0).
- Includes rules to ensure each color is used only once and appears in only one position.
- Adds clues and restrictions, such as “blue is not in position 0.”
- Analyzes the information to deduce the possible color arrangement.

src > mastermind.py > ...

```python
from logic import *

colors = ["red", "blue", "green", "yellow"]
symbols = []
for i in range(4):
    for color in colors:
        symbols.append(Symbol(f"{color}{i}"))

knowledge = And()

# Each color has a position.
for color in colors:
    knowledge.add(Or(
        Symbol(f"{color}0"),
        Symbol(f"{color}1"),
        Symbol(f"{color}2"),
        Symbol(f"{color}3")
    ))

# Only one position per color.
for color in colors:
    for i in range(4):
        for j in range(4):
            if i != j:
                knowledge.add(Implication(
                    Symbol(f"{color}{i}"), Not(Symbol(f"{color}{j}"))
                ))

# Only one color per position.
for i in range(4):
    for c1 in colors:
        for c2 in colors:
            if c1 != c2:
                knowledge.add(Implication(
                    Symbol(f"{c1}{i}"), Not(Symbol(f"{c2}{i}"))
                ))
```

```python
knowledge.add(Or(
    And(Symbol("red0"), Symbol("blue1"), Not(Symbol("green2")), Not(Symbol("yellow3"))),
    And(Symbol("red0"), Symbol("green2"), Not(Symbol("blue1")), Not(Symbol("yellow3"))),
    And(Symbol("red0"), Symbol("yellow3"), Not(Symbol("blue1")), Not(Symbol("green2"))),
    And(Symbol("blue1"), Symbol("green2"), Not(Symbol("red0")), Not(Symbol("yellow3"))),
    And(Symbol("blue1"), Symbol("yellow3"), Not(Symbol("red0")), Not(Symbol("green2"))),
    And(Symbol("green2"), Symbol("yellow3"), Not(Symbol("red0")), Not(Symbol("blue1")))
))

knowledge.add(And(
    Not(Symbol("blue0")),
    Not(Symbol("red1")),
    Not(Symbol("green2")),
    Not(Symbol("yellow3"))
))

for symbol in symbols:
    if model_check(knowledge, symbol):
        print(symbol)
```

Result :

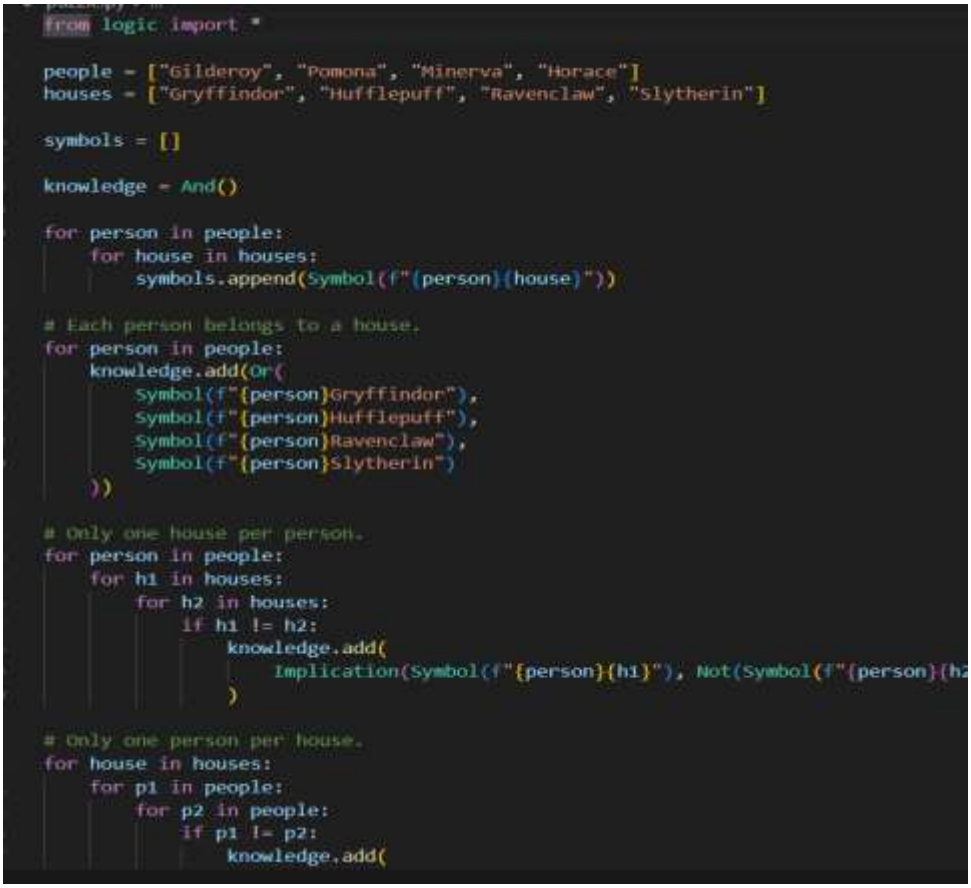
```
PS C:\Users\mmd14\OneDrive\Desktop\knowledge Representation> &
entation/src/mastermind.py"
red0
blue1
yellow2
green3
PS C:\Users\mmd14\OneDrive\Desktop\knowledge Representation>
```

puzzle.py:

The program solves a logic puzzle about assigning people to houses, like figuring out who is in Gryffindor.

How It Works:

- Uses symbols like *GilderoyGryffindor* to show possible house assignments.
- Adds rules to make sure each person belongs to one house, and each house has only one person.
- Includes known facts and determines what house assignments can be figured out.



```
from logic import *

people = ["Gilderoy", "Pomona", "Minerva", "Horace"]
houses = ["Gryffindor", "Hufflepuff", "Ravenclaw", "Slytherin"]

symbols = []

knowledge = And()

for person in people:
    for house in houses:
        symbols.append(Symbol(f"{person}{house}"))

# Each person belongs to a house.
for person in people:
    knowledge.add(Or(
        Symbol(f"{person}Gryffindor"),
        Symbol(f"{person}Hufflepuff"),
        Symbol(f"{person}Ravenclaw"),
        Symbol(f"{person}Slytherin")
    ))

# Only one house per person.
for person in people:
    for h1 in houses:
        for h2 in houses:
            if h1 != h2:
                knowledge.add(
                    Implication(Symbol(f"{person}{h1}"), Not(Symbol(f"{person}{h2
                ))

# Only one person per house.
for house in houses:
    for p1 in people:
        for p2 in people:
            if p1 != p2:
                knowledge.add(
```

```
                knowledge.add(
                    Implication(Symbol(f"{p1}{house}"), Not(Symbol(f"{p2}{house}")))
                )

knowledge.add(
    Or(Symbol("GilderoyGryffindor"), Symbol("GilderoyRavenclaw"))
)

knowledge.add(
    Not(Symbol("PomonaSlytherin"))
)

knowledge.add(
    Symbol("MinervaGryffindor")
)

for symbol in symbols:
    if model_check(knowledge, symbol):
        print(symbol)
```

Result :

```
PS C:\Users\mmd14\OneDrive\Desktop\knowledge Representation>
entation/src/puzzle.py"
GilderoyRavenclaw
PomonaHufflepuff
MinervaGryffindor
HoraceSlytherin
PS C:\Users\mmd14\OneDrive\Desktop\knowledge Representation>
```

How It Works:

- Symbols: Represent facts, like *ColMustard* for a suspect or *red0* for red in position 0.
- Logic Sentences: Combine symbols with rules like "If A, then B" or "A and B."
- Knowledge Base: A collection of rules and known facts.
- Model Checking: The program tests different combinations to see what the facts and rules imply (e.g., if *ColMustard* could be the suspect).

Simplified Example:

- Suppose we know:
    - "If it rains, the grass is wet."
    - "The grass is wet."
- The program represents these statements as symbols and uses logic to determine whether it must be raining.

Key Points:

- These programs translate real-world problems into logical rules and symbols for reasoning.
- Model checking automates the process of determining what is true or possible based on the given rules and facts. Complex problems can use First-Order Logic for variables and relationships.