EX.NO.: 08

DATE: 24.01.2025

ANALYSE SENTENCE

To represent sentences involving relationships, properties, and quantifiers in first-order logic (FOL) using symbolic notations and predicates.

PROCEDURE:

1. Identify the Key Entities
    a. Extract individuals, objects, or subjects mentioned in the sentences.
    b. For example:
        i. Angus, Cyril, Irene (f1).
        ii. Tofu, Bertie (f2).
2. Define Predicates
    a. Identify relationships or properties between the entities.
    b. Create symbolic functions to represent predicates.
        i. Example predicates:
            1. `Likes(x, y)`: Represents "x likes y."
            2. `Hates(x, y)`: Represents "x hates y."
            3. `Taller(x, y)`: Represents "x is taller than y."
3. Construct Logical Expressions
    a. Translate the sentences into logical expressions using the predicates.
    b. Use logical operators like ∧ (AND), ∨ (OR), ¬ (NOT), and quantifiers (if applicable).
    c. Examples:
        i. "Angus likes Cyril and Irene hates Cyril" becomes:
           `Likes(Angus, Cyril) ∧ Hates(Irene, Cyril)`
        ii. "Tofu is taller than Bertie" becomes:
           `Taller(Tofu, Bertie)`
4. Handle Quantifiers (if necessary)
    a. If the sentence involves words like "someone," "everyone," or "nobody," represent them with quantifiers:
        i. ∃ (Exists): "There exists" or "someone."
        ii. ∀ (ForAll): "For all" or "everyone."
5. Ensure Consistency
    a. Confirm that the logical expressions are aligned with the intended meaning of the sentences.
    b. Double-check the use of logical operators and quantifiers.
6. Output the Translations

CODE AND OUTPUT

```
from sympy import symbols, Not, And, Or, Implies

# Define propositional variables
A, B, C, D, R, S, O, T, H, Y, M, I = symbols("A B C D R S O T H Y M I")

# Translations
sentences = {
    "a": Implies(A, Not(B)),                        # If Angus sings, Bertie doesn't
sulk
    "b": And(C, D),                                 # Cyril runs and barks
    "c": Implies(Not(R), S),                        # It will snow if it doesn’t rain
    "d": Not(Implies(Or(O, T), H)),                 # Irene will not be happy if Olive
or Tofu comes
    "e": And(Not(C), Not(S)),                       # Pat didn’t cough or sneeze
```

```
    "f": Implies(Implies(Not(Y), Not(M)), Implies(Not(C), Not(I))),  # Conditional
statement
}

# Print and verify the expressions
for key, sentence in sentences.items():
    print(f"{key}: {sentence}")
```

```
a: Implies(A, ~B)
b: C & D
c: Implies(~R, S)
d: ~(Implies(O | T, H))
e: ~C & ~S
f: Implies(Implies(~Y, ~M), Implies(~C, ~I))
```

```
# Define entities
from sympy import symbols

# Entities (individuals)
a, b, c, i, t, o, j, p, m = symbols("a b c i t o j p m")

# Define predicates as strings
Likes = lambda x, y: f"Likes({x}, {y})"
Hates = lambda x, y: f"Hates({x}, {y})"
Taller = lambda x, y: f"Taller({x}, {y})"
Loves = lambda x, y: f"Loves({x}, {y})"
Saw = lambda x, y: f"Saw({x}, {y})"
FourLeggedFriend = lambda x: f"FourLeggedFriend({x})"
Near = lambda x, y: f"Near({x}, {y})"

# Example sentences
f1 = f"{Likes('Angus', 'Cyril')} ∧ {Hates('Irene', 'Cyril')}"  # Angus likes Cyril and
Irene hates Cyril
f2 = f"{Taller('Tofu', 'Bertie')}"                              # Tofu is taller than
Bertie
f3 = f"{Loves('Bruce', 'Bruce')} ∧ {Loves('Pat', 'Pat')}"       # Bruce loves himself
and Pat does too
f4 = f"{Saw('Cyril', 'Bertie')} ∧ ¬{Saw('Angus', 'Bertie')}"    # Cyril saw Bertie,
Angus didn't
f5 = f"{FourLeggedFriend('Cyril')}"                             # Cyril is a four-legged
friend
f6 = f"{Near('Tofu', 'Olive')}"                                 # Tofu and Olive are
near each other

# Print sentences
for i, f in enumerate([f1, f2, f3, f4, f5, f6], start=1):
    print(f"f{i}: {f}")
```

```
f1: Likes(Angus, Cyril) ∧ Hates(Irene, Cyril)
f2: Taller(Tofu, Bertie)
f3: Loves(Bruce, Bruce) ∧ Loves(Pat, Pat)
f4: Saw(Cyril, Bertie) ∧ ¬Saw(Angus, Bertie)
f5: FourLeggedFriend(Cyril)
f6: Near(Tofu, Olive)
```

```
from sympy import symbols, And, Or, Not, Implies
```

```
# Define entities
a, b, c, i, t, o, j, p, m = symbols("a b c i t o j p m")  # Individuals

# Define predicates as symbolic representations
Likes = lambda x, y: f"Likes({x}, {y})"
SmilesAt = lambda x, y: f"SmilesAt({x}, {y})"
Coughs = lambda x: f"Coughs({x})"
Sneezes = lambda x: f"Sneezes({x})"

# Example quantified sentences represented as strings
f7 = f"∃x ({Likes('Angus', 'x')})"                # Angus likes someone
f8 = f"∃x ({Likes('Angus', 'x')} ∧ {Likes('x', 'Angus')})"  # Angus loves a dog who loves him
f9 = f"¬∃x ({SmilesAt('x', 'Pat')})"              # Nobody smiles at Pat
f10 = f"∃x ({Coughs('x')} ∧ {Sneezes('x')})"      # Somebody coughs and sneezes
f11 = f"¬∃x ({Coughs('x')} ∨ {Sneezes('x')})"     # Nobody coughed or sneezed

# Print quantified formulas
print(f"f7: {f7}")
print(f"f8: {f8}")
print(f"f9: {f9}")
print(f"f10: {f10}")
print(f"f11: {f11}")
```

```
f7: ∃x (Likes(Angus, x))
f8: ∃x (Likes(Angus, x) ∧ Likes(x, Angus))
f9: ¬∃x (SmilesAt(x, Pat))
f10: ∃x (Coughs(x) ∧ Sneezes(x))
f11: ¬∃x (Coughs(x) ∨ Sneezes(x))
```

```
# Define λ-abstraction functions
def lambda_abstraction(action, *args):
    return f"λ {', '.join(args)}. {action}({', '.join(args)})"

# Examples
abs1 = lambda_abstraction("Feed", "x", "c") + " ∧ " + lambda_abstraction("Give", "x", "Cappuccino", "a")
abs2 = lambda_abstraction("Given", "'War and Peace'", "x", "p")
abs3 = lambda_abstraction("∀ y (Loves(y, x))", "x")
abs4 = lambda_abstraction("∀ y (Loves(y, x) ∨ Detests(y, x))", "x")
abs5 = lambda_abstraction("∀ y (Loves(y, x)) ∧ ∀ z (¬Detests(z, x))", "x")

# Print abstractions
for i, abs in enumerate([abs1, abs2, abs3, abs4, abs5], start=1):
```

```python
    print(f"λ-Abstraction {i}: {abs}")
```

```
λ-Abstraction 1: λ x, c. Feed(x, c) ∧ λ x, Cappuccino, a. Give(x, Cappuccino, a)
λ-Abstraction 2: λ 'War and Peace', x, p. Given('War and Peace', x, p)
λ-Abstraction 3: λ x. ∀ y (Loves(y, x))(x)
λ-Abstraction 4: λ x. ∀ y (Loves(y, x) ∨ Detests(y, x))(x)
λ-Abstraction 5: λ x. ∀ y (Loves(y, x)) ∧ ∀ z (¬Detests(z, x))(x)
```