FOL to CNF Algorithm:

11) Converting FOL into CNF

Input first_order_logic_statement:

Eliminate implications: Replace $(A \rightarrow B)$ with $(\neg A \vee B)$

Move $\neg$ (negations) inward using De Morgan's Law

Standardize variables: Ensure each quantifier has unique name.

Move Quantifiers to the front (prenex form)

Skolomize: Eliminate existential quantifiers to introducing skolem functions.

Drop Universal Quantifiers

Distribute $\vee$ over $\wedge$ to obtain CNF form

Output CNF clauses

Output:

Original statement: (A & B) → C

CNF form: ~A | ~B | C

Code:

```
from sympy import symbols, Not, Or, And, Implies, Equivalent
from sympy.logic.boolalg import to_cnf

def fol_to_cnf(fol_expr):
    fol_expr = fol_expr.replace(Equivalent, lambda a, b: And(Implies(a, b),
Implies(b, a)))
    fol_expr = fol_expr.replace(Implies, lambda a, b: Or(Not(a), b))
    cnf_form = to_cnf(fol_expr, simplify=True)
    return cnf_form

def main():
    P = symbols("P")
    Q = symbols("Q")
    R = symbols("R")

    fol_expr1 = Implies(P, Q)
    print("Example 1: P → Q")
    print("Original FOL Expression:")
    print(fol_expr1)

    cnf1 = fol_to_cnf(fol_expr1)
    print("\nCNF Form:")
    print(cnf1)

    fol_expr2 = Implies(Or(P, Not(Q)), Or(Q, R))
    print("\nExample 2: (P V ¬Q) → (Q V R)")
    print("Original FOL Expression:")
    print(fol_expr2)

    cnf2 = fol_to_cnf(fol_expr2)
    print("\nCNF Form:")
    print(cnf2)
```

```python
if __name__ == "__main__":
    main()
```

Output:

Example 1: P → Q

Original FOL Expression:

Implies(P, Q)

CNF Form:

Q | ~P

Example 2: (P ∨ ¬Q) → (Q ∨ R)

Original FOL Expression:

Implies(P | ~Q, Q | R)

CNF Form:

Q | R