Lab program - 9

Convert a first order logic to conjunctive normal form

Algorithm

Step 1: eliminate implications
Replace implications with their equivalent forms using the rule $p \Rightarrow q \equiv \neg p \vee q$

Step 2: Move negations inwards
Apply DE Morgan's law to move negation inwards.
For example
$\neg(p \wedge q) \equiv \neg p \vee \neg q$
$\neg(p \vee q) \equiv \neg p \wedge \neg q$

Step 3: Distribute disjunctions over conjunctions
- Distribute disjunctions ($\vee$) over conjunctions ($\wedge$) using the distributive property.
For example $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

Step 4: convert to CNF form
Ensure that formula is in CNF by applying any additional simplifications if needed

Code

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

def eliminate_implications(formula):
    return formula.subs(Implies(p, q),
                        Or(Not(p), q))

def move_negations_inwards(formula):
    return formula.simplify()

def distribute_disjunctions_over_conjunctions
                  (formula):
    return formula.simplify()

def fol_to_cnf(fol_formula):
    formula_step1 = eliminate_implications
                    (fol_formula)

    formula_step2 = move_negations_inwards
                    (formula_step1)

    formula_step3 = distribute_disjunctions_
                    over_conjunctions
                    (formula_step2)

    cnf_formula = to_cnf(formula_
                  step3)

    return cnf_formula
```

```
user_input = input ("Enter a first order
                    logic formula:")
p, q, r = symbols ('p q r')
fol_formula = eval (user_input, {'p': p,
                    'q': q, 'r': r})

cnf_formula = fol_to_cnf (fol_formula)
print ("\nFOL Formula", fol_formula)
print ("CNF Formula:", cnf_formula)
```

Output

```
Enter a first order logic : p & (~q | r)
FOL formula : And(p, Or(Not(q), r))
CNF Formula : Or(And(p, r), And
                 (p, ~q))
∨(∧(p, r) ∧(p, ~q)
```