Problem Set 3.

Xuanqi Wei
1009353209
Q5

(a). Let $p \in \mathcal{F}$.

Let $P(p)$: "$T(p)$ is logically equivalent to p and has negation only on variables, $N(p)$ is logically equivalent to $\neg p$ and has negation only on variables."

WTS: $\forall p \in \mathcal{F}, P(p)$.

I'll apply structural induction on p.

Base Case: Let $i \in \mathbb{N}$.

WTS: $P(x_i)$ holds.

By definition of T, gives $T(x_i) = x_i$, which the claim $T(x_i)$ is logically equivalent to x_i. Since there's no negation, the claim that $T(x_i)$ has negation only on variables is vacuously true.

By definition of N, gives $N(x_i) = x_i$, which the claim $N(x_i)$ is logically equivalent to x_i. Since there's no negation, the claim that $N(x_i)$ has negation only on variables is vacuously true.

Induction Step. Let $p, q \in \mathcal{F}$.

Induction Hypothesis: Assume $P(p)$ and $P(q)$.

WTS: ① $P((p \wedge q))$ ② $P((p \vee q))$ ③ $P((\neg p))$.

To prove $P((p \wedge q))$, I'll prove both $T((p \wedge q))$ and $N((p \wedge q))$ only have negation on variables.

For ①: $T((p \wedge q)) = T(p) \wedge T(q)$ (by definition).

$= p \wedge q$ (by I.H).

$N((p \wedge q)) = N(p) \vee N(q)$ (by definition).

$= (\neg p) \vee (\neg q)$. (by I.H). $= \neg (p \wedge q)$ (by De Morgan's Law).

To prove $P((p \vee q))$, I'll prove both $T((p \vee q))$ and $N((p \vee q))$ only have negation on variables.

For ②: $T((p \vee q)) = T(p) \vee T(q)$ (by definition).

$= p \vee q$ (by I.H).

$N((p \vee q)) = N(p) \wedge N(q)$ (by definition).

$= (\neg p) \wedge (\neg q)$. (by I.H) $= \neg(p \vee q)$ (by De Morgan's Law)

To prove $P((\neg p))$, I'll prove both $T((\neg p))$ and $N((\neg p))$ only have negation on variables.

For ③: $T((\neg p)) = N(p)$. (by definition).

By I.H. we know $N(p)$ is logically equivalent to $\neg p$ and only has negation on variables, gives $T((\neg p))$ is logically equivallent to $\neg p$ and only has negation on variables.

$N((\neg p)) = T(p)$. (by definition).

By I.H. we know $T(p)$ is logically equivalent to p and only has negation on variables, gives $N((\neg p))$ is logically equivallent to $p = \neg(\neg p)$. and only has negation on variables.

Therfore. $P(p)$ is true as needed. ∎

(b).

```python
def t(form):
    """
    Writing a recursive Python function to compute T
    """
    # Considering for x_i
    if type(form) is str:
        return form

    # Considering for (p, "and", q) or (p, "or", q)
    if len(form) == 3:
        (p, op, q) = form
        # Since we recursively call T on p and q individually, and the number of operators that f has is one more than
        # the number of operators that p and q have together. Thus, I've made each recursive has fewer than before
        return t(p), op, t(q)

    # Considering ("not", y)
    else:
        (_, f2) = form
        # ("not", ("not", p))
        if len(f2) == 2:
            (_, p) = f2
            # I will recursively call T on p and p has two fewer operators than f now
            return t(p)

        # Considering ("not", (p, "and", q)) or ("not", (p, "or", q))
        if len(f2) == 3:
            (p, op, q) = f2
            # I will split the cases into "and" or "or"
            if op == "and":
                # I will recursively call T. Since p and q is separate from the "and" and op, p and q have at least two
                # fewer operators than f. Thus, ("not", p) and ("not", q) have at least one less operator than f has.
                return t(("not", p)), "or", t(("not", q))
            # op == "or"
            else:
                # I will recursively call T. Since p and q is separate from the "and" and op, p and q have at least two
                # fewer operators than f. Thus, ("not", p) and ("not", q) have at least one less operator than f has.
                return t(("not", p)), "and", t(("not", q))
        else:
            return form
```