Quiz Section 1: Propositional Logic Translation

Task 1 – Equivalences

Prove that each of the following equivalences using an equivalence chain.

a) $(p \to \neg p) \wedge (\neg p \to p) \equiv \mathsf{F}$

b) $\neg p \to (q \to r) \equiv q \to (p \vee r)$

c) $\neg(q \vee p) \vee \neg(r \vee \neg p) \to p \equiv p \vee q$

d) $p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
Hint: You may use the definition of biconditional that $p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$.

Task 2 – Welcome to the $\forall \mathrm{LL} \exists n$ School!

Let the domain of discourse be all students and classes at UW. Define the predicates $\mathsf{CS}(x)$ to mean that x majors in CS (presumably x is a student) and $\mathsf{CE}(x)$ to mean that x majors in CE (presumably x is a student). Define the predicates $\mathsf{CSE}(y)$ to mean that y is a CSE class (presumably y is a class) and $\mathsf{MATH}(y)$ to mean that y is a MATH class (presumably y is a class). Define the predicate $\mathsf{Wants}(x, y)$ to mean that x wants to take y (presumably x is a student and y is a class), the predicate $\mathsf{Likes}(x, y)$ to mean that x likes y (presumably x is a student and y is a class), and the predicate $\mathsf{HasToTake}(x, y)$ to mean that x has to take y (presumably x is a student and y is a class).

Translate each of the following logical statements into English. You should not simplify. However, you should use the techniques shown in lecture for producing more natural translations when restricting domains and for avoiding the introduction of variable names when not necessary.

a) $\neg \exists x \, (\mathsf{CS}(x) \wedge \mathsf{CE}(x))$

b) $\exists x \, \big(\mathsf{CS}(x) \wedge \exists y \, (\mathsf{CSE}(y) \wedge \neg \mathsf{HasToTake}(x, y) \wedge \mathsf{Likes}(x, y))\big)$

c) $\forall x \, \big(\mathsf{CE}(x) \rightarrow \exists y \, (\mathsf{MATH}(y) \wedge \mathsf{HasToTake}(x, y))\big)$

d) $\exists x \, \big((\mathsf{CS}(x) \vee \mathsf{CE}(x)) \wedge \forall y \, (\mathsf{CSE}(y) \rightarrow \mathsf{Wants}(x, y))\big)$

Task 3 – Translate to Logic

Express each of these system specifications using predicates, quantifiers, and logical connectives. For some of these problems, more than one translation will be reasonable depending on your choice of predicates.

a) Every user has access to an electronic mailbox.

b) The system mailbox can be accessed by everyone in the group if the file system is locked.

c) The firewall is in a diagnostic state only if the proxy server is in a diagnostic state.

d) At least one router is functioning normally if the throughput is between 100kbps and 500 kbps and the proxy server is not in diagnostic mode.

Task 4 – The Calm Before the Form

The Java project you are working on contains the following function. It takes four boolean arguments, a, b, c, and d, and calculates some boolean function of them called E:

```
public static boolean E(boolean a, boolean b, boolean c, boolean d) {
  if (!(a || b))
    return false;
  if (!(!a || !b))
    return false;
  if (!(a || c))
    return false;
  return (b || !d);
}
```

The code checks the value of the four disjunctions $a \vee b$, $\neg a \vee \neg b$, $a \vee c$, and $b \vee \neg d$ and returns true iff *all four* of them are true. In other words, it calculates the conjunction (and) of those four expressions. Specifically, it calculates the same value as the following (non-canonical) CNF expression:

$$(a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee c) \wedge (b \vee \neg d)$$

Note that this Java code could be written equivalently as a single `return` statement:

```
return (a || b) && (!a || !b) && (a || c) && (b || !d);
```

The Java compiler would automatically translate that into the series of `if` statements above that returns `false` as soon as one disjunction is found to evaluate as false, a process known as "short circuiting".

a) Write a truth table for E. Include columns for a, b, c, d, all four disjunctions, and E.

b) Write the **canonical** DNF expression for E.

c) Translate your DNF expression into a new Java implementation of `E`.

Like above, your code should be a series of `if` statements and then a `return`, except that, now, each `if` should check the value of a conjunction and return `true` if it is true.

(Do not simplify. Translate your expression to code in the most direct way possible.)