COMS10014 Solutions 3: Logic 2

1. Truth Tables and Terminology

1. $p \to (q \to p)$ is a tautology:

p	q	$q \to p$	$p \to (q \to p)$
T	T	T	T
T	F	T	T
F	T	F	T
F	F	T	T

2. $\neg p \vee (\neg p \to q)$ is a tautology:

p	q	$\neg p$	$\neg p \to q$	$\neg p \vee (\neg p \to q)$
T	T	F	T	T
T	F	F	T	T
F	T	T	T	T
F	F	T	F	T

3. $(p \vee q) \to (p \wedge q)$ is a contingency:

p	q	$p \vee q$	$p \wedge q$	$(p \vee q) \to (p \wedge q)$
T	T	F	F	T
T	F	T	F	F
F	T	T	F	F
F	F	T	T	T

4. $(p \wedge q) \to (p \vee q)$ is a tautology:

p	q	$p \vee q$	$p \wedge q$	$(p \wedge q) \to (p \vee q)$
T	T	F	F	T
T	F	T	F	T
F	T	T	F	T
F	F	T	T	T

5. $(p \vee q) \wedge (p \vee \neg q) \wedge (\neg p \vee q) \wedge (\neg p \vee \neg q)$ is a contradiction, as each row has one clause that is F:

p	q	$p \vee q$	$p \vee \neg q$	$\neg p \vee q$	$\neg p \vee \neg q$	$\wedge$ of all
T	T	F	T	T	T	F
T	F	T	T	F	T	F
F	T	T	F	T	T	F
F	F	T	T	T	F	F

6. $p \vee (q \rightarrow p)$ is a contingency:

p	q	$q \rightarrow p$	$p \vee (q \rightarrow p)$
T	T	T	T
T	F	T	T
F	T	F	F
F	F	T	T

7. $(p \rightarrow q) \vee (q \rightarrow p)$ is a tautology:

p	q	$p \rightarrow q$	$q \rightarrow p$	$(p \rightarrow q) \vee (q \rightarrow p)$
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

8. $\neg(p \rightarrow q) \wedge p$ is a contingency:

p	q	$p \rightarrow q$	$\neg(p \rightarrow q)$	$\neg(p \rightarrow q) \wedge p$
T	T	T	F	F
T	F	F	T	T
F	T	T	F	F
F	F	T	F	F

b. Satisfiable means the same as ‘not a contradiction’, so 1,2,3,4,6,7,8 are satisfiable; 5 isd not.

2. Implications

1. If we have frost tonight, then I won’t cycle in tomorrow.
   Converse: If I don’t cycle in tomorrow, then we will have frost tonight.
   Inverse: If we don’t have frost tonight, then I will cycle in tomorrow.
   Contrapositive: If I cycle in tomorrow, then we will not have frost tonight.

2. My cat comes in whenever it is hungry.
   Converse: If my cat comes in, then it is hungry.
   Inverse: If my cat is not hungry, then it does not come in.
   Contrapositive: If my cat does not come in, then it is not hungry.

3. When you hear the fire alarm, you need to vacate the building.
   Converse: If you need to vacate the building, you will hear the fire alarm.
   Inverse: If you don’t hear the fire alarm, you don’t need to vacate the building.
   Contrapositive: If you don’t need to vacate the building, then you won’t hear the fire alarm.

4. If it is hot tomorrow, then we will go swimming.
   Converse: If we go swimming tomorrow, then it will be hot.
   Inverse: If it is not hot tomorrow, then we will not go swimming.
   Contrapositive: If we don't go swimming, then it will not be hot tomorrow.

5. People who don't pay their tax by the deadline will be fined.
   Converse: If you are fined, then you haven't paid your tax by the deadline.
   Inverse: If you pay your tax by the deadline, then you will not be fined.
   Contrapositive: If you are not fined, then you have paid your tax by the deadline.

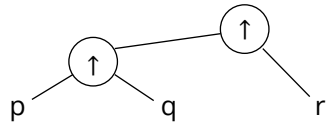
3. NAND

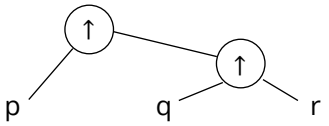
1.

p	q	$p \uparrow q$
T	T	F
T	F	T
F	T	T
F	F	T

2. Yes. Reversing the operands does not make a difference: $T \uparrow F \equiv F \uparrow T$ (and the other cases are clear as both sides are the same anyway).

3. The question here is whether $(p \uparrow q) \uparrow r \equiv p \uparrow (q \uparrow r)$. Let's do a truth table:





p	q	r				
T	T	T	F	**T**	**T**	F
T	T	F	F	**T**	**F**	T
T	F	T	T	**F**	**F**	T
T	F	F	T	**T**	**F**	T
F	T	T	T	**F**	**T**	F
F	T	F	T	**T**	**T**	T
F	F	T	T	**F**	**T**	T
F	F	F	T	**T**	**T**	T

The two shaded columns are not the same, therefore the operation is not associative.

4. Here we calculate

	$p \to \neg q$	given
$\equiv$	$\neg p \vee \neg q$	from $a \to b \equiv \neg a \vee b$
$\equiv$	$\neg(p \wedge q)$	DeMorgan
$\equiv$	$p \uparrow q$	definition of ↑

5. This is a case of expanding $p \wedge q \equiv (p \uparrow q) \uparrow (p \uparrow q)$ twice in a row, giving

$$\Big(p \uparrow \big((q \uparrow r) \uparrow (q \uparrow r)\big)\Big) \uparrow \Big(p \uparrow \big((q \uparrow r) \uparrow (q \uparrow r)\big)\Big)$$

6. Here we have some work to do:

$$
\begin{array}{lll}
 & \neg(p \oplus q) & \\
\equiv & \neg\big((p \vee q) \wedge \neg(p \wedge q)\big) & \text{Definition of } \oplus \\
\equiv & \neg\big((p \vee q) \wedge (p \uparrow q)\big) & \text{Definition of } \uparrow \\
\equiv & (p \vee q) \uparrow (p \uparrow q) & \text{Definition of } \uparrow \\
\equiv & \neg\neg(p \vee q) \uparrow (p \uparrow q) & \text{Double negation introduction} \\
\equiv & \neg(\neg p \wedge \neg q) \uparrow (p \uparrow q) & \text{DeMorgan} \\
\equiv & (\neg p \uparrow \neg q) \uparrow (p \uparrow q) & \text{Definition of } \uparrow \\
\equiv & \big((p \uparrow p) \uparrow (q \uparrow q)\big) \uparrow (p \uparrow q) & \neg p \equiv p \uparrow p
\end{array}
$$

The strategy here is first of all to replace any negated ∧ we find with an ↑. When we are left with an ∨, the standard way to change it into an ∧ is to introduce a double negation followed by DeMorgan to 'consume' one of the negations again; conveniently enough this leaves us with a negated ∧ which is exactly what we need.

4. Normal Forms

1. For the first proof, we have to do the proof 'from scratch' as there is nothing we can build on yet. This means giving a method that, for any term A, produces an equivalent term B using only the allowed operations. In principle, A could contain implications and NAND/NOR and any other binary operations that we have not named yet, but we do not care as we work with a truth table: the proof is semantic. (The term B produced this way happens to have the extra property that it is in a normal form, which is useful for hardware design.)

For the second proof, we already know that the set from a. is functionally complete, so we do not have to reason about every term A but only terms A' already in DNF, since we could turn any term A into an equivalent one in DNF with the argument from the previous proof. All we have to show is that we can replace ∨ operations using ∧ and ¬. This proof is syntactic.

2. Given a truth table,

1. Form a truth table for the formula.
2. Compute a clause for each row, as for DNF.
3. Compute the negation of the clause and apply DeMorgan, to get a clause with the literals reversed and union instead of intersection operations. For example, the row $a = T, b = T, c = F$ would become $(\neg a \vee \neg b \vee c)$. This is the extra step.
4. Take the intersection of these clauses, for the rows where the negated formula is *false*.

This procedure must produce a formula in CNF, as it produced an intersection of clauses of unions of literals. But why is this the correct formula?

Two facts:

1. A CNF formula is true if and only if *all* of its clauses are satisfied by the same assignment, unlike a DNF formula where it is enough to satisfy one clause.
2. The DNF clause for each row in the truth table is true for the assignment in that row, and false for all other rows. The CNF clause for each row, being the negation of the DNF clause, is false in its own row, and true in all other rows.

(For example, for the row $a = T, b = T, c = F$ the DNF clause is $a \wedge b \wedge \neg c$ which is true for this one assignment and false for the other 7 assignments. The CNF clause is $\neg a \vee \neg b \vee c$ which is false for this assignment, and true for the other 7.)

- For any row where the original formula is false, by 2. the CNF clause for this row will be false, and therefore by 1. the whole CNF formula will be false.
- For any row where the original formula is true, all the CNF clauses will be true. This is because each clause is only false in its 'own' row, but we only took the clauses for rows where the original formula was false.

This shows that the formula constructed this way is in CNF, and equivalent to the original one.

3. The definition of CNF is [for DNF, swap ∧ and ∨]:

1. A formula made of a list of one or more clauses, combined with ∧.
2. Each clause is one or more literals, combined with ∨.
3. Each literal is either a variable, or the negation of a variable.

A list of only one clause 'combined with ∧', as read by a mathematician, simply means a clause with no ∧ operators at all, as there is nothing to combine. For the same reason, a sum of only one number does not need a + sign. Similarly, a clause with only one literal needs no ∨.

Aisha is correct that the formula is in CNF, as you can interpret it as a single clause with three literals, combined with ∨. Brenda is correct that the formula is in DNF, as you can interpret it as three clauses with a single literal each.

4. Here we have:

1. Is both in CNF (two clauses of one literal each) and in DNF (one clause of two literals).
2. Is both in CNF and DNF (in both cases, one clause of one literal).
3. Same as 1.
4. Is both in CNF (one clause of two literals) and in DNF (two clauses of one literal each).
5. Neither CNF nor DNF, in both normal forms negations can only appear immediately before variables.
6. Same as 5.
7. Same as 5.
8. Is in DNF (two clauses of two variables each) but not in CNF (in CNF, a ∨ node can only appear within clauses, so it cannot have a ∧ child).

5. a. The mistake is that nothing in the definition of CNF says a clause can only contain a variable once, so for example $(a \vee \neg b \vee \neg a)$ is a valid clause. The statement that 'each clause "encodes" a row of a truth table' is not correct. Our procedures do always produce formulas with this property, but a procedure that turns CNF to DNF *in general* cannot assume this.

b. Take a formula with n literals that is only false in one case, such as $a_1 \vee a_2 \vee \ldots \vee a_n$ (this is a valid CNF formula consisting of only one clause, as we have established above). Applying our procedure would not only create a truth table with 2^n rows, but the resulting DNF formula would have $(2^n - 1)$ clauses, each with n literals, to cover all the other rows. This is an exponential increase in the size of the formula, which is not efficient in either the informal or in the technical sense of the word.