Lesson Review: Module Summary

Response to and Request for feedback

I would love some feedback on if I have covered everything in Boolean algebra. I think its un-necessary to include a explanation of every law (as they are seen in my examples) I had some trouble understanding brackets (however this was due to NUMBAS not displaying brackets, so I didn't think they were important)

Module Learning Objectives

1. recognise and translate atomic and compound propositions
2. construct truth tables
3. apply laws of Boolean algebra.

Summarising the content:

Propositions: A proposition is a statement that can only evaluate too only either true or false. Propositions are denoted by symbols, for example: *a, b, y, z.* Propositions are very important in the world of Computer Science as there are many instructions that can be evaluated to either true or false. **A proposition does not have to have its output known** for example, *q* "I will be an Olympic swimmer". Yes, it unlikely (I don't even like swimming), but it is either *true* OR *false*. Saying this, we can identify statements that are not propositions, for example, *g* "how are you". Yes, it can evaluate to good or bad, but it can't be true or false. True and false is often also referred to 0 or 1. We can observe that this knowledge satisfies module learning objective one: "recognise and translate atomic and compound propositions".

Algebra of Propositions: An atomic proposition refers to a statement in which cannot be broken down into smaller statements for example, $a \wedge b$. You can see that this cannot be broken down any smaller (it's at its base form). On the other hand, a compound proposition are multiple atomic propositions stringed together using connectives. **Connectives** refer to the actions that combine atomic propositions, some connectives include of: ¬ (said as 'not'), ∨ (said as 'or'), and ∧ (said as 'and'). ¬ essentially means the opposite. You can think of it as how a negative number on a number line and how it is a opposite reflection. In other words, $\neg p$ is only true when *p* is false. ∧ is only true when both *p* and *q* are true, if either statement is false, it will be false. ∨ is true when either *p* or *q* are true, saying this, if both are false, the statement is false. There are also more complex connectives such as XOR, NAND, and NOR which are interesting, however are beyond the scope of this unit. Building on our knowledge of connectives, we also have → which is read as "implies". **This arrow symbol empresses the relationship between two statements**, for example, $p \rightarrow q$ means that based on *p*, *q* is the next logical sequence of events. Similarly, ↔ is read as "if and only if", is true if and only if both propositions have the same logical value. In other words, it is highlighting the relationship between the two propitiations. **Expanding on implications, the way the arrow points does not matter, however, what matters is what proposition is a sequence of the other, sometimes the arrow can point both ways, but sometimes it can't.** For examples if we take *p* 'is prime' and *q* 'is 7' we can easily identify that *q implies p*, however, *p does not imply q*. There are some cases where *P* and *Q* are logically equivalent, this means that both *P* and *Q* have the same truth value in every possible situation. We can use truth tables to determine whether they have the same truth value given every possible situation. For example, the double negation laws states that ¬ ¬ P has the same logical equivalence as P. If we create one truth tables for each proposition, we will see they always have the same truth result. When looking at a law such as double negation, we don't actually have to create a table to comprehend this as we can think about a –(-n) where n is a number to just be the positive flipside. This topic expands of our knowledge of module learning objective one "recognise and translate atomic and compound propositions".

Truth Tables: Sometimes, it can **be very complex to deduce what a very large compound proposition is doing**, especially if it has a lot of connectives, this is where a truth table comes in handy. **A truth table is a way of evaluating every possible combination (and therefor output) of an expression.** Just say we have too propositions, *p* and *q*, using a truth table, we can find out all the

combinations of true and false between these two expressions and then evaluate the connectives and therefor the outcomes. It is helpful when creating a compound proposition from a worded problem. We can see this topic satisfies module learning objective two: “construct truth tables”. You can see lots of examples of this in my notes provided.

Boolean Algebra: If there are many atomic propositions and connectives, it may be time consuming and computationally expensive to create truth tables as based on the number of propositions, the number of combinations increases linearly (2^*n*). **This is where simplifying the expression before creating a truth table comes in handy.** I will not explore each law in a summary, but in short, the laws simplify the expression based on relationships between the propositions and connectives. One important thing to consider when exploring Boolean algebra is that brackets are important. For examples when we see the compound proposition $q \vee (\neg p \wedge q)$ the $\neg p$ and q are $\wedge$’ed together followed by being $\vee$’d by q. This topic satisfies module learning objective three: “apply laws of Boolean algebra.” *Lots* of examples of Boolean algebra is shown in my notes provided.

How is this useful: Boolean Algebra and Truth tables are important as they provide a low-level understanding of how a computer commutes specific instructs. I have noticed that this specific topic is being explored in my SIT111 Computer systems class, highlighting it significance and importance in Computer Sciences as a whole.

Reflecting on the content:

- What is the most important thing you learnt in this module?

The most important thing I learnt in this module was what propositions are, and how they are used within computers to evaluate statements to be either true or false. In general, I think this is a very important topic within Computer Science as a whole rather than just Discrete mathematics.

- How does this relate to what you already know?

From my SIT111 class, we explored abstraction, this gave me an infighting into what could be happening in computers in my everyday life. For example, a fridge has propositional logic determining if the light is turned on or the temperature is too cold.

- Why do you think your course team wants you to learn the content of this module for your degree?

I think the course team wants students to learn this content as it develops a deeper understanding of how computer working. This assists in creating software in SIT102 as coding would be impossible without Boolean logic.

MY NOTES: