# Math for CS 2015/2019 solutions to "In-Class Problems Week 2, Fri. (Session 4)"

https://github.com/spamegg1

October 19, 2022

## Contents



## 1 Problem 1

Prove by truth table that OR distributes over AND, namely,

P OR (Q AND R) is equivalent to (P OR Q) AND (P OR R)

*Proof.* I will use $\wedge$ for AND, $\vee$ for OR.

| $P$ | $Q$ | $R$ | $Q \wedge R$ | $P \vee Q$ | $P \vee R$ | $P \vee (Q \wedge R)$ | $(P \vee Q) \wedge (P \vee R)$ |
|---|---|---|---|---|---|---|---|
| $T$ | $T$ | $T$ | $T$ | $T$ | $T$ | $T$ | $T$ |
| $T$ | $T$ | $F$ | $F$ | $T$ | $T$ | $T$ | $T$ |
| $T$ | $F$ | $T$ | $F$ | $T$ | $T$ | $T$ | $T$ |
| $T$ | $F$ | $F$ | $F$ | $T$ | $T$ | $T$ | $T$ |
| $F$ | $T$ | $T$ | $T$ | $T$ | $T$ | $T$ | $T$ |
| $F$ | $T$ | $F$ | $F$ | $T$ | $F$ | $F$ | $F$ |
| $F$ | $F$ | $T$ | $F$ | $F$ | $T$ | $F$ | $F$ |
| $F$ | $F$ | $F$ | $F$ | $F$ | $F$ | $F$ | $F$ |

□

# 2 Problem 2

This problem examines whether the following specifications are satisfiable:

1. If the file system is not locked, then

(a) new messages will be queued.

(b) new messages will be sent to the messages buffer.

(c) the system is functioning normally, and conversely, if the system is functioning normally, then the file system is not locked.

2. If new messages are not queued, then they will be sent to the messages buffer.

3. New messages will not be sent to the message buffer.

## 2.1 (a)

Begin by translating the five specifications into propositional formulas using four propositional variables:

L ::= file system locked;

Q ::= new messages are queued;

B ::= new messages are sent to the message buffer;

N ::= system functioning normally.

*Proof.* 1(a) is translated as: **NOT(L) IMPLIES Q**.

1(b) is translated as: **NOT(L) IMPLIES B**.

1(c) is translated as: **NOT(L) IFF N**.

2 is translated as: **NOT(Q) IMPLIES B**.

3 is translated as: **NOT(B)**. □

## 2.2 (b)

Demonstrate that this set of specifications is satisfiable by describing a single truth assignment for the variables L, Q, B, N and verifying that under this assignment, all the specifications are true.

*Proof.* (Remember how IMPLIES works: **true IMPLIES false** is false, and the other 3 possibilities for IMPLIES are all true.)

I will use T for true, F for false.

Consider the truth assignment **L: T, Q: T, B: F, N: F**. Then:

**NOT(L) IMPLIES Q** is: **NOT(T) IMPLIES T**, which is: **F IMPLIES T**, which is true.

**NOT(L) IMPLIES B** is: **NOT(T) IMPLIES F**, which is: **F IMPLIES F**, which is true.

**NOT(L) IFF N** is: **NOT(T) IFF F**, which is: **F IFF F**, which is true.

**NOT(Q) IMPLIES B** is: **NOT(T) IMPLIES F**, which is: **F IMPLIES F**, which is true.

**NOT(B)** is: **NOT(F)**, which is true. □

### 2.3 (c)

Argue that the assignment determined in part (b) is the only one that does the job.

*Proof.* To satisfy 3: **NOT(B)**, B has to be false. That's the only option.

With B set to false, 2 becomes: **NOT(Q) IMPLIES F**. Due to the way IMPLIES works, the only way to satisfy this is to make **NOT(Q)** false. So Q has to be true.

With B set to false, 1(b) becomes: **NOT(L) IMPLIES F**. By a similar argument, L has to be true.

With L set to true, 1(c) becomes: **F IFF N**, which forces N to be false.

So, all 4 variables are forced to have their truth values in order to satisfy all the statements. □

## 3 Problem 3

Propositional logic comes up in digital circuit design using the convention that T corresponds to 1 and F to 0. A simple example is a 2-bit half-adder circuit. This circuit has 3 binary inputs, $a_1, a_0$ and $b$, and 3 binary outputs, $c, s_1, s_0$. The 2-bit word $a_1a_0$ gives the binary representation of an integer, $k$, between 0 and 3. The 3-bit word $cs_1s_0$ gives the binary representation of $k + b$. The third output bit, $c$, is called the final carry bit.

So if $k$ and $b$ were both 1, then the value of $a_1a_0$ would be 01 and the value of the output $cs_1s_0$ would be 010, namely, the 3-bit binary representation of $1 + 1$.

In fact, the final carry bit equals 1 only when all three binary inputs are 1, that is, when $k = 3$ and $b = 1$.

In that case, the value of $cs_1s_0$ is 100, namely, the binary representation of $3 + 1$.

This 2-bit half-adder could be described by the following formulas:

$$\begin{aligned}
c_0 &= b \\
s_0 &= a_0 \text{ XOR } c_0 \\
c_1 &= a_0 \text{ AND } c_0 && \text{the carry into column 1} \\
s_1 &= a_1 \text{ XOR } c_1 \\
c_2 &= a_1 \text{ AND } c_1 && \text{the carry into column 2} \\
c &= c_2
\end{aligned}$$

## 3.1 (a)

Generalize the above construction of a 2-bit half-adder to an $n+1$ bit half-adder with inputs $a_n, \ldots, a_1, a_0$ and $b$ and outputs $c, s_n, \ldots, s_1, s_0$. That is, give simple formulas for $s_i$ and $c_i$ for $0 \leq i \leq n+1$, where $c_i$ is the carry into column $i+1$, and $c = c_{n+1}$.

*Proof.* The $n+1$-bit word $a_n \ldots a_1 a_0$ will be the binary representation of an integer, $s$, between 0 and $2^{n+1} - 1$. The circuit will have $n+2$ outputs $c, o_n, \ldots, o_1, o_0$ where the $n+2$-bit word $co_n \ldots o_1 o_0$ gives the binary representation of $s + b$.

Here are some simple formulas that define such a half-adder:

$$\begin{aligned}
c_0 &= b \\
o_i &= a_i \text{ XOR } c_i && \text{for } 0 \leq i \leq n \\
c_{i+1} &= a_i \text{ AND } c_i && \text{for } 0 \leq i \leq n \\
c &= c_{n+1}
\end{aligned}$$

□

## 3.2 (b)

Write similar definitions for the digits and carries in the sum of two $n+1$-bit binary numbers $a_n \ldots a_1 a_0$ and $b_n \ldots b_1 b_0$.

*Proof.* Define

$$\begin{aligned}
c_0 &= 0 \\
o_i &= a_i \text{ XOR } b_i \text{ XOR } c_i && \text{for } 0 \leq i \leq n \\
c_{i+1} &= (a_i \text{ AND } b_i) \text{ OR } (a_i \text{ AND } c_i) \text{ OR } (b_i \text{ AND } c_i) && \text{for } 0 \leq i \leq n \\
c &= c_{n+1}
\end{aligned}$$

□

Visualized as digital circuits, the above adders consist of a sequence of single-digit half-adders or adders strung together in series. These circuits mimic ordinary pencil-and-paper addition, where a carry into a column is calculated directly from the carry into the previous column, and the carries have to ripple across all the columns before the carry into the final column is determined. Circuits with this design are called ripple-carry adders. Ripple-carry adders are easy to understand and remember and require a nearly minimal number of operations. But the higher-order output bits and the final carry take time proportional to n to reach their final values.

### 3.3 (c)

How many of each of the propositional operations does your adder from part (b) use to calculate the sum?

*Proof.* The scheme given in the solution to part (b) uses $3(n+1)$ AND's, $2(n+1)$ XOR's, and $2(n+1)$ OR's for a total of $7(n+1)$ operations. Because $c_0$ is always 0, you could skip all the operations involving it. Then the counts are $3n+1$ AND's, $2n+1$ XOR's, and $2n$ OR's for a total of $7n+2$ operations. □

## 4 Problem 4

When the mathematician says to his student, "If a function is not continuous, then it is not differentiable," then letting D stand for "differentiable" and C for continuous, the only proper translation of the mathematician's statement would be

NOT(C) IMPLIES NOT(D)

or equivalently,

D IMPLIES C.

But when a mother says to her son, "If you don't do your homework, then you can't watch TV," then letting T stand for "can watch TV" and H for "do your homework," a reasonable translation of the mother's statement would be

NOT(H) IFF NOT(T)

or equivalently,

H IFF T.

Explain why it is reasonable to translate these two IF-THEN statements in different ways into propositional formulas.

*Proof.* We know that a differentiable function must be continuous, so when a function is not continuous, it is also not differentiable. Now Mathematicians use IMPLIES in the technical way given by its truth table. In particular, if a function is continuous then to a Mathematician, the implication

is automatically true since the hypothesis (left hand side of the IMPLIES) is false. So whether or not continuity holds, the Mathematician could comfortably assert the IMPLIES statement knowing it is correct.

And of course a Mathematician does not mean IFF, since she knows a function that is not differentiable may well be continuous.

On the other hand, while the Mother certainly means that her son cannot watch TV if he does not do his homework, both she and her son most likely understand that if he does do his homework, then he will be allowed watch TV. In this case, even though the Mother uses an IF-THEN phrasing, she really means IFF.

On the other hand, circumstances in the household might be that the boy may watch TV when he has not only done his homework, but also cleaned up his room. In this case, just doing homework would not imply being allowed to watch TV –the boy won't be allowed to watch TV if he hasn't cleaned his room, even if he has done his homework.

The general point here is that semantics (meaning) trumps syntax (sentence structure): even though the Mathematician's and Mother's statements have the same structure, their meaning may warrant different translations into precise logical language. □