

## In-Class Problems Week 2, Mon.

### Problem 1.

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

$$P \text{ OR } (Q \text{ AND } R) \text{ is equivalent to } (P \text{ OR } Q) \text{ AND } (P \text{ OR } R) \quad (1)$$

### Problem 2.

This problem<sup>1</sup> 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.

(a) Begin by translating the five specifications into propositional formulas using the 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.

(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.


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

### Problem 3.

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

$$\text{NOT}(C) \text{ IMPLIES } \text{NOT}(D),$$

---

 2017, Albert R Meyer. This work is available under the terms of the [Creative Commons Attribution-ShareAlike 3.0 license](https://creativecommons.org/licenses/by-sa/3.0/).

<sup>1</sup>Revised from Rosen, 5th edition, Exercise 1.1.36

or equivalently,

$$D \text{ IMPLIES } C.$$

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

$$\text{NOT}(H) \text{ IFF } \text{NOT}(T),$$

or equivalently,

$$H \text{ IFF } T.$$

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

**Problem 4. (a)** Verify by truth table that

$$(P \text{ IMPLIES } Q) \text{ OR } (Q \text{ IMPLIES } P)$$

is valid.

(b) Let  $P$  and  $Q$  be propositional formulas. Describe a single formula  $R$  using only AND’s, OR’s, NOT’s, and copies of  $P$  and  $Q$ , such that  $R$  is valid iff  $P$  and  $Q$  are equivalent.

(c) A propositional formula is *satisfiable* iff there is an assignment of truth values to its variables—an *environment*—that makes it true. Explain why

$$P \text{ is valid iff NOT}(P) \text{ is not satisfiable.}$$

(d) A set of propositional formulas  $P_1, \dots, P_k$  is *consistent* iff there is an environment in which they are all true. Write a formula  $S$  such that the set  $P_1, \dots, P_k$  is *not* consistent iff  $S$  is valid.

### Problem 5.

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 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:

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

(a) Generalize the above construction of a 2-bit half-adder to an  $n+1$  bit half-adder with inputs  $a_n, \dots, a_1, a_0$  and  $b$  and outputs  $c, s_n, \dots, s_1, s_0$ . 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  $c = c_{n+1}$ .

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

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.

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