Which answer in this list is the correct answer to *this* question?

William John Holden

November 26, 2019

An intriguing post on StackExchange (`https://math.stackexchange.com/q/2217248/474318`) asks "which answer in this list is the correct answer to this question?"

1. All of the below.
2. None of the below.
3. All of the above.
4. One of the above.
5. None of the above.
6. None of the above.

I will call two ways to solve this problem the "analytical" approach and the "computational" approach.

1 Analytical Approach

First, we have to understand the *implication* operator ($\implies$) used in mathematical logic. Consider the *statement*, "if it rains, then I will wear a jacket." Is the statement still true if I am wearing a jacket and it is not raining? Yes. Maybe it is snowing, and "if it snows, then I will wear a jacket" is also a "true" statement.

What if it is raining, but I do not have my jacket? Then the original statement is false; I claimed that I would wear a jacket if it rained, but did not.

A stronger cousin to the implication operator is the *biconditional* ($\iff$). In a sentence, the biconditional operator often sounds like "if and only if." For example, "if and only if it rains, then I will bring an umbrella." The prefix *bi* hints that the biconditional operator is actually two implications together: "if it rains then I will an umbrella, and I will only have an umbrella if it rains." Symbolically,

$$(A \implies B) \land (B \implies A) \iff (A \iff B).$$

In many texts, you might see the equals ($=$) or equivalent ($\equiv$) symbols used interchangably with $\iff$.

The biconditional operator is what we need to solve this problem using mathematical logic. The following formulas restate the problem in symbols using this operator. The symbol $\wedge$ means "and," the symbol $\vee$ means "or" (inclusive), and the symbol $\neg$ means "not."

$$
\begin{aligned}
1 &\iff (2 \wedge 3 \wedge 4 \wedge 5 \wedge 6) \\
2 &\iff (\neg 3 \wedge \neg 4 \wedge \neg 5 \wedge \neg 6) \\
3 &\iff (1 \wedge 2) \\
4 &\iff (1 \wedge \neg 2 \wedge \neg 3) \vee (\neg 1 \wedge 2 \wedge \neg 3) \vee (\neg 1 \wedge \neg 2 \wedge 3) \\
5 &\iff (\neg 1 \wedge \neg 2 \wedge \neg 3 \wedge \neg 4) \\
6 &\iff (\neg 1 \wedge \neg 2 \wedge \neg 3 \wedge \neg 4 \wedge \neg 5)
\end{aligned}
$$

The fourth statement deserves some special attention. This statement is a "gadget" that is true when exactly one of the three variables is true.

A seventh constraint is implied in the question text: only one of the six statements is true. The "exactly-one-of" gadget in disjunctive normal form (DNF) for six variables contains six clauses, each falsifying all but one variable:

$$
\begin{aligned}
7 \iff\ & (1 \wedge \neg 2 \wedge \neg 3 \wedge \neg 4 \wedge \neg 5 \wedge \neg 6) \vee (\neg 1 \wedge 2 \wedge \neg 3 \wedge \neg 4 \wedge \neg 5 \wedge \neg 6) \vee \\
& (\neg 1 \wedge \neg 2 \wedge 3 \wedge \neg 4 \wedge \neg 5 \wedge \neg 6) \vee (\neg 1 \wedge \neg 2 \wedge \neg 3 \wedge 4 \wedge \neg 5 \wedge \neg 6) \vee \\
& (\neg 1 \wedge \neg 2 \wedge \neg 3 \wedge \neg 4 \wedge 5 \wedge \neg 6) \vee (\neg 1 \wedge \neg 2 \wedge \neg 3 \wedge \neg 4 \wedge \neg 5 \wedge 6)
\end{aligned}
$$

We will not actually need 7, but it is useful to recognize non-obvious rules implied in a problem statement. We will now prove the correct answer by construction.

1.1 Statement 1

$$
\begin{aligned}
1 &\implies 2 \wedge 1 \implies 3 \\
2 &\implies \neg 3 \text{ (contradiction)}
\end{aligned}
$$

1.2 Statement 2

For this one, we need to step through carefully. Assume statement 2 is true. If statement 3 is false (only 2 is true) then 1 must be false. If 1 and 3 are false and 2 is true then 4 is also true, which does not satisfy 2.

$$
\begin{aligned}
2 &\implies \neg 3 \\
\neg 3 &\implies \neg(1 \wedge 2) \iff \neg 1 \vee \neg 2 \text{ (DeMorgan's Law)} \\
2 \wedge (\neg 2 \vee \neg 1) &\implies \neg 1 \\
\neg 1 \wedge 2 \wedge \neg 3 &\implies 4 \\
4 &\implies \neg 2 \text{ (contradiction)}
\end{aligned}
$$

1.3 Statement 3

$$
\begin{aligned}
3 &\implies 2 \\
2 &\implies \neg 3 \text{ (contradiction)}
\end{aligned}
$$

1.4 Statement 4

In the above three sections we have already shown $\neg 1$, $\neg 2$, and $\neg 3$. Statement 4 states that exactly one of these three is true, which we now know is not possible.

$$\neg 1 \wedge \neg 2 \wedge \neg 3 \implies \neg 4$$

1.5 Statement 5

We have proven $\neg 1$, $\neg 2$, $\neg 3$, and $\neg 4$. This means that statement 5 is true.

$$\neg 1 \wedge \neg 2 \wedge \neg 3 \wedge \neg 4 \implies 5$$

1.6 Statement 6

You might have been tempted to guess statement 6, but now that we know statement 5 is true, statement 6 must be false.

$$5 \implies \neg 6$$

2 Computational Approach

Computing machines are well-suited to solve this problem. Problems like this are tedious and difficult to solve. Human intuition (hunches) can be distracting. There is a simple and guaranteed technique to solving logic problems of this nature: try every possibility. Unfortunately, there are 2^n possible assignments of literals to a Boolean formula of n variables.

Writing out a truth table does precisely this. We explicitly write out one column per variable, form all 2^n combinations of variables, and show the outcome of each hypothesis.

Constraint solvers describe a class of *domain-specific* programming languages. z3 (`https://github.com/Z3Prover/z3`) is a solver by Microsoft Research which can quickly solve difficult problems (or prove that no solution can exist). z3's syntax is similar to a Lisp. You can practice with z3 at `https://rise4fun.com/z3/`.

The following program models the question posed in this paper. Paste the below source code into z3 or click `https://rise4fun.com/Z3/Vorh` and click the triangle button.

```
(declare-const a Bool)
(declare-const b Bool)
(declare-const c Bool)
(declare-const d Bool)
(declare-const e Bool)
(declare-const f Bool)

(assert (= a (and b c d e f)))
(assert (= b (and (not c) (not d) (not e) (not f))))
(assert (= c (and a b)))
(assert (= d (or
  (and a (not b) (not c))
  (and (not a) b (not c))
  (and (not a) (not b) c))))
(assert (= e (and (not a) (not b) (not c) (not d))))
(assert (= f (and (not a) (not b) (not c) (not d) (not e))))
(assert (or a b c d e f))

(check-sat)
(get-model)
```

z3 should output `sat`, meaning the formula is satisfiable. The `get-model` function displays the literal assignments for each variable needed to satisfy the formula.

```
sat
(model
  (define-fun f () Bool
    false)
  (define-fun b () Bool
    false)
  (define-fun a () Bool
    false)
  (define-fun c () Bool
    false)
  (define-fun d () Bool
    false)
  (define-fun e () Bool
    true)
)
```