Sesh 4: May 2, 2024

3.1 Propositions from Propositions

In English, we can modify, combine, and relate propositions with words such as "not," "and," "or," "implies," and "if-then." For example, we can combine three propositions into one like this:

If all humans are mortal **and** all Greeks are human, **then** all Greeks are mortal.

For the next while, we won't be much concerned with the internals of propositions—whether they involve mathematics or Greek mortality—but rather with how propositions are combined and related. So, we'll frequently use variables such as P and Q in place of specific propositions such as "All humans are mortal" and "$2 + 3 = 5$." The understanding is that these *propositional variables*, like propositions, can take on only the values **T** (true) and **F** (false). Propositional variables are also called *Boolean variables* after their inventor, the nineteenth century mathematician George—you guessed it—Boole.

3.1.1 NOT, AND, and OR

Mathematicians use the words NOT, AND, and OR for operations that change or combine propositions. The precise mathematical meaning of these special words can be specified by *truth tables*. For example, if P is a proposition, then so is "NOT(P)," and the truth value of the proposition "NOT(P)" is determined by the truth value of P according to the following truth table:

→ they are mathematically defined by these truth tables, so accept them w/o much concern on whether it's intuitive or not.

P	NOT(P)
T	**F**
F	**T**

The first row of the table indicates that when proposition P is true, the proposition "NOT(P)" is false. The second line indicates that when P is false, "NOT(P)" is true. This is probably what you would expect.

In general, a truth table indicates the true/false value of a proposition for each possible set of truth values for the variables. For example, the truth table for the proposition "P AND Q" has four lines, since there are four settings of truth values for the two variables:

P	Q	P AND Q
T	**T**	**T**
T	**F**	**F**
F	**T**	**F**
F	**F**	**F**

According to this table, the proposition "P AND Q" is true only when P and Q are both true. This is probably the way you ordinarily think about the word "and."

There is a subtlety in the truth table for "P OR Q":

P	Q	P OR Q
T	**T**	**T**
T	**F**	**T**
F	**T**	**T**
F	**F**	**F**

The first row of this table says that "P OR Q" is true even if *both* P and Q are true. This isn't always the intended meaning of "or" in everyday speech, but this is the standard definition in mathematical writing. So if a mathematician says, "You may have cake, or you may have ice cream," he means that you *could* have both.

If you want to exclude the possibility of having both cake *and* ice cream, you should combine them with the *exclusive-or* operation, XOR:

P	Q	P XOR Q
T	**T**	**F**
T	**F**	**T**
F	**T**	**T**
F	**F**	**F**

3.1.2 IMPLIES

The combining operation with the least intuitive technical meaning is "implies." Here is its truth table, with the lines labeled so we can refer to them later.

P	Q	P IMPLIES Q	
T	**T**	**T**	(tt)
T	**F**	**F**	(tf)
F	**T**	**T**	(ft)
F	**F**	**T**	(ff)

The truth table for implications can be summarized in words as follows:

An implication is true exactly when the if-part is false or the then-part is true.

This sentence is worth remembering; a large fraction of all mathematical statements are of the if-then form!

Let's experiment with this definition. For example, is the following proposition true or false?

"If Goldbach's Conjecture is true, then $x^2 \geq 0$ for every real number x."

Now, we already mentioned that no one knows whether Goldbach's Conjecture, Proposition 1.1.8, is true or false. But that doesn't prevent you from answering the question! This proposition has the form P IMPLIES Q where the *hypothesis*, P, is "Goldbach's Conjecture is true" and the *conclusion*, Q, is "$x^2 \geq 0$ for every real number x." Since the conclusion is definitely true, we're on either line (tt) or line (ft) of the truth table. Either way, the proposition as a whole is *true*!

One of our original examples demonstrates an even stranger side of implications.

"If pigs fly, then you can understand the Chebyshev bound."

Don't take this as an insult; we just need to figure out whether this proposition is true or false. Curiously, the answer has *nothing* to do with whether or not you can understand the Chebyshev bound. Pigs do not fly, so we're on either line (ft) or line (ff) of the truth table. In both cases, the proposition is *true*!

In contrast, here's an example of a false implication:

"If the moon shines white, then the moon is made of white cheddar."

Yes, the moon shines white. But, no, the moon is not made of white cheddar cheese. So we're on line (tf) of the truth table, and the proposition is false.

False Hypotheses

It often bothers people when they first learn that implications which have false hypotheses are considered to be true. But implications with false hypotheses hardly ever come up in ordinary settings, so there's not much reason to be bothered by whatever truth assignment logicians and mathematicians choose to give them.

There are, of course, good reasons for the mathematical convention that implications are true when their hypotheses are false. An illustrative example is a system specification (see Problem 3.12) which consisted of a series of, say, a dozen rules,

> if C_i: the system sensors are in condition i, then A_i: the system takes action i,

or more concisely,

$$C_i \text{ IMPLIES } A_i$$

for $1 \leq i \leq 12$. Then the fact that the system obeys the specification would be expressed by saying that the AND

$$[C_1 \text{ IMPLIES } A_1] \text{ AND } [C_2 \text{ IMPLIES } A_2] \text{ AND } \cdots \text{ AND } [C_{12} \text{ IMPLIES } A_{12}] \quad (3.1)$$

of these rules was always true.

For example, suppose only conditions C_2 and C_5 are true, and the system indeed takes the specified actions A_2 and A_5. This means that in this case the system is behaving according to specification, and accordingly we want the formula (3.1) to come out true. Now the implications C_2 IMPLIES A_2 and C_5 IMPLIES A_5 are both true because both their hypotheses and their conclusions are true. But in order for (3.1) to be true, we need all the other implications with the false hypotheses C_i for $i \neq 2, 5$ to be true. This is exactly what the rule for implications with false hypotheses accomplishes.

3.1.3 If and Only If

Mathematicians commonly join propositions in one additional way that doesn't arise in ordinary speech. The proposition "P if and only if Q" asserts that P and Q have the same truth value. Either both are true or both are false.

P	Q	P IFF Q
T	**T**	**T**
T	**F**	**F**
F	**T**	**F**
F	**F**	**T**

For example, the following if-and-only-if statement is true for every real number x:

$$x^2 - 4 \geq 0 \text{ IFF } |x| \geq 2.$$

For some values of x, *both* inequalities are true. For other values of x, *neither* inequality is true. In every case, however, the IFF proposition as a whole is true.

3.2 Propositional Logic in Computer Programs

Propositions and logical connectives arise all the time in computer programs. For example, consider the following snippet, which could be either C, C++, or Java:

```
if ( x > 0 || (x <= 0 && y > 100) )
    ⋮
(further instructions)
```

Java uses the symbol `||` for "OR," and the symbol `&&` for "AND." The *further instructions* are carried out only if the proposition following the word `if` is true. On closer inspection, this big expression is built from two simpler propositions.

Let A be the proposition that `x > 0`, and let B be the proposition that `y > 100`. Then we can rewrite the condition as

$$A \text{ OR } (\text{NOT}(A) \text{ AND } B). \tag{3.2}$$

3.2.1 Truth Table Calculation

A truth table calculation reveals that the more complicated expression 3.2 always has the same truth value as

$$A \text{ OR } B. \tag{3.3}$$

We begin with a table with just the truth values of A and B:

A	B	A OR (NOT(A) AND B)	A OR B
T	**T**		
T	**F**		
F	**T**		
F	**F**		

These values are enough to fill in two more columns:

A	B	A OR (NOT(A) AND B)	A OR B
T	**T**	**F**	**T**
T	**F**	**F**	**T**
F	**T**	**T**	**T**
F	**F**	**T**	**F**

Now we have the values needed to fill in the AND column:

A	B	A OR (NOT(A) AND B)	A OR B
T	**T**	**F** **F**	**T**
T	**F**	**F** **F**	**T**
F	**T**	**T** **T**	**T**
F	**F**	**T** **F**	**F**

and this provides the values needed to fill in the remaining column for the first OR:

A	B	A OR (NOT(A) AND B)	A OR B
T	**T**	**T** **F** **F**	**T**
T	**F**	**T** **F** **F**	**T**
F	**T**	**T** **T** **T**	**T**
F	**F**	**F** **T** **F**	**F**

Expressions whose truth values always match are called *equivalent*. Since the two emphasized columns of truth values of the two expressions are the same, they are equiv.

equivalent. So we can simplify the code snippet without changing the program's behavior by replacing the complicated expression with an equivalent simpler one:

```
if ( x > 0 || y > 100 )
    ⋮
(further instructions)
```

The equivalence of (3.2) and (3.3) can also be confirmed reasoning by cases:

A is **T**. An expression of the form (**T** OR anything) is equivalent to **T**. Since *A* is **T** both (3.2) and (3.3) in this case are of this form, so they have the same truth value, namely, **T**.

A is **F**. An expression of the form (**F** OR *anything*) will have same truth value as *anything*. Since *A* is **F**, (3.3) has the same truth value as *B*.

An expression of the form (**T** AND *anything*) is equivalent to *anything*, as is any expression of the form **F** OR *anything*. So in this case *A* OR (NOT(*A*) AND *B*) is equivalent to (NOT(*A*) AND *B*), which in turn is equivalent to *B*.

Therefore both (3.2) and (3.3) will have the same truth value in this case, namely, the value of *B*.

Simplifying logical expressions has real practical importance in computer science. Expression simplification in programs like the one above can make a program easier to read and understand. Simplified programs may also run faster, since they require fewer operations. In hardware, simplifying expressions can decrease the number of logic gates on a chip because digital circuits can be described by logical formulas (see Problems 3.5 and 3.6). Minimizing the logical formulas corresponds to reducing the number of gates in the circuit. The payoff of gate minimization is potentially enormous: a chip with fewer gates is smaller, consumes less power, has a lower defect rate, and is cheaper to manufacture.

3.2.2 Cryptic Notation

Java uses symbols like "&&" and "||" in place of AND and OR. Circuit designers use "·" and "+," and actually refer to AND as a product and OR as a sum. Mathematicians use still other symbols, given in the table below.

English	Symbolic Notation
NOT(P)	$\neg P$ (alternatively, $\overline{P}$)
P AND Q	$P \wedge Q$
P OR Q	$P \vee Q$
P IMPLIES Q	$P \longrightarrow Q$
if P then Q	$P \longrightarrow Q$
P IFF Q	$P \longleftrightarrow Q$
P XOR Q	$P \oplus Q$

For example, using this notation, "If P AND NOT(Q), then R" would be written:

$$(P \wedge \overline{Q}) \longrightarrow R.$$

The mathematical notation is concise but cryptic. Words such as "AND" and "OR" are easier to remember and won't get confused with operations on numbers. We will often use $\overline{P}$ as an abbreviation for NOT(P), but aside from that, we mostly stick to the words—except when formulas would otherwise run off the page.

3.3 Equivalence and Validity

3.3.1 Implications and Contrapositives

Do these two sentences say the same thing?

If I am hungry, then I am grumpy.
If I am not grumpy, then I am not hungry.

We can settle the issue by recasting both sentences in terms of propositional logic. Let P be the proposition "I am hungry" and Q be "I am grumpy." The first sentence says "P IMPLIES Q" and the second says "NOT(Q) IMPLIES NOT(P)." Once more, we can compare these two statements in a truth table:

P	Q	(P IMPLIES Q)	(NOT(Q)	IMPLIES	NOT(P))
T	**T**	**T**	**F**	**T**	**F**
T	**F**	**F**	**T**	**F**	**F**
F	**T**	**T**	**F**	**T**	**T**
F	**F**	**T**	**T**	**T**	**T**

Sure enough, the highlighted columns showing the truth values of these two statements are the same. A statement of the form "NOT(Q) IMPLIES NOT(P)" is called

Three axioms that don't directly correspond to number properties are

$$A \text{ AND } A \longleftrightarrow A \qquad \text{(idempotence for AND)}$$
$$A \text{ AND } \overline{A} \longleftrightarrow \mathbf{F} \qquad \text{(contradiction for AND)} \tag{3.9}$$
$$\text{NOT}(\overline{A}) \longleftrightarrow A \qquad \text{(double negation)} \tag{3.10}$$

It is associativity (3.8) that justifies writing A AND B AND C without specifying whether it is parenthesized as A AND $(B$ AND $C)$ or $(A$ AND $B)$ AND C. Both ways of inserting parentheses yield equivalent formulas.

There are a corresponding set of equivalences for OR which we won't bother to list, except for the OR rule corresponding to contradiction for AND (3.9):

$$A \text{ OR } \overline{A} \longleftrightarrow \mathbf{T} \qquad \text{(validity for OR)}$$

Finally, there are *DeMorgan's Laws* which explain how to distribute NOT's over AND's and OR's:

$$\text{NOT}(A \text{ AND } B) \longleftrightarrow \overline{A} \text{ OR } \overline{B} \qquad \text{(DeMorgan for AND)} \tag{3.11}$$
$$\text{NOT}(A \text{ OR } B) \longleftrightarrow \overline{A} \text{ AND } \overline{B} \qquad \text{(DeMorgan for OR)} \tag{3.12}$$

All of these axioms can be verified easily with truth tables.

These axioms are all that's needed to convert any formula to a disjunctive normal form. We can illustrate how they work by applying them to turn the negation of formula (3.5),

$$\text{NOT}((A \text{ AND } B) \text{ OR } (A \text{ AND } C)). \tag{3.13}$$

into disjunctive normal form.

We start by applying DeMorgan's Law for OR (3.12) to (3.13) in order to move the NOT deeper into the formula. This gives

$$\text{NOT}(A \text{ AND } B) \text{ AND NOT}(A \text{ AND } C).$$

Now applying Demorgan's Law for AND (3.11) to the two innermost AND-terms, gives

$$(\overline{A} \text{ OR } \overline{B}) \text{ AND } (\overline{A} \text{ OR } \overline{C}). \tag{3.14}$$

At this point NOT only applies to variables, and we won't need Demorgan's Laws any further.

Now we will repeatedly apply The Distributivity of AND over OR (Theorem 3.4.1) to turn (3.14) into a disjunctive form. To start, we'll distribute $(\overline{A}$ OR $\overline{B})$ over AND to get

$$((\overline{A} \text{ OR } \overline{B}) \text{ AND } \overline{A}) \text{ OR } ((\overline{A} \text{ OR } \overline{B}) \text{ AND } \overline{C}).$$

Using distributivity over both AND's we get

$$((\overline{A} \text{ AND } \overline{A}) \text{ OR } (\overline{B} \text{ AND } \overline{A})) \text{ OR } ((\overline{A} \text{ AND } \overline{C}) \text{ OR } (\overline{B} \text{ AND } \overline{C})).$$

By the way, we've implicitly used commutativity (3.7) here to justify distributing over an AND from the right. Now applying idempotence to remove the duplicate occurrence of $\overline{A}$ we get

$$(\overline{A} \text{ OR } (\overline{B} \text{ AND } \overline{A})) \text{ OR } ((\overline{A} \text{ AND } \overline{C}) \text{ OR } (\overline{B} \text{ AND } \overline{C})).$$

Associativity now allows dropping the parentheses around the terms being OR'd to yield the following disjunctive form for (3.13):

$$\overline{A} \text{ OR } (\overline{B} \text{ AND } \overline{A}) \text{ OR } (\overline{A} \text{ AND } \overline{C}) \text{ OR } (\overline{B} \text{ AND } \overline{C}). \tag{3.15}$$

The last step is to turn each of these AND-terms into a disjunctive normal form with all three variables A, B, and C. We'll illustrate how to do this for the second AND-term $(\overline{B} \text{ AND } \overline{A})$. This term needs to mention C to be in normal form. To introduce C, we use validity for OR and identity for AND to conclude that

$$(\overline{B} \text{ AND } \overline{A}) \longleftrightarrow (\overline{B} \text{ AND } \overline{A}) \text{ AND } (C \text{ OR } \overline{C}).$$

Now distributing $(\overline{B} \text{ AND } \overline{A})$ over the OR yields the disjunctive normal form

$$(\overline{B} \text{ AND } \overline{A} \text{ AND } C) \text{ OR } (\overline{B} \text{ AND } \overline{A} \text{ AND } \overline{C}).$$

Doing the same thing to the other AND-terms in (3.15) finally gives a disjunctive normal form for (3.5):

$$\begin{array}{l}
(\overline{A} \text{ AND } B \text{ AND } C) \text{ OR } (\overline{A} \text{ AND } B \text{ AND } \overline{C}) \text{ OR} \\
(\overline{A} \text{ AND } \overline{B} \text{ AND } C) \text{ OR } (\overline{A} \text{ AND } \overline{B} \text{ AND } \overline{C}) \text{ OR} \\
(\overline{B} \text{ AND } \overline{A} \text{ AND } C) \text{ OR } (\overline{B} \text{ AND } \overline{A} \text{ AND } \overline{C}) \text{ OR} \\
(\overline{A} \text{ AND } \overline{C} \text{ AND } B) \text{ OR } (\overline{A} \text{ AND } \overline{C} \text{ AND } \overline{B}) \text{ OR} \\
(\overline{B} \text{ AND } \overline{C} \text{ AND } A) \text{ OR } (\overline{B} \text{ AND } \overline{C} \text{ AND } \overline{A}).
\end{array}$$

Using commutativity to sort the term and OR-idempotence to remove duplicates, finally yields a unique sorted DNF:

$$\begin{array}{l}
(A \text{ AND } \overline{B} \text{ AND } \overline{C}) \text{ OR} \\
(\overline{A} \text{ AND } B \text{ AND } C) \text{ OR} \\
(\overline{A} \text{ AND } B \text{ AND } \overline{C}) \text{ OR} \\
(\overline{A} \text{ AND } \overline{B} \text{ AND } C) \text{ OR} \\
(\overline{A} \text{ AND } \overline{B} \text{ AND } \overline{C}).
\end{array}$$

This example illustrates a strategy for applying these equivalences to convert any formula into disjunctive normal form, and conversion to conjunctive normal form works similarly, which explains:

Theorem 3.4.4. *Any propositional formula can be transformed into disjunctive normal form or a conjunctive normal form using the equivalences listed above.*

What has this got to do with equivalence? That's easy: to prove that two formulas are equivalent, convert them both to disjunctive normal form over the set of variables that appear in the terms. Then use commutativity to sort the variables and AND-terms so they all appear in some standard order. We claim the formulas are equivalent iff they have the same sorted disjunctive normal form. This is obvious if they do have the same disjunctive normal form. But conversely, the way we read off a disjunctive normal form from a truth table shows that two different sorted DNF's over the same set of variables correspond to different truth tables and hence to inequivalent formulas. This proves

Theorem 3.4.5 (Completeness of the propositional equivalence axioms)**.** *Two propositional formula are equivalent iff they can be proved equivalent using the equivalence axioms listed above.*

The benefit of the axioms is that they leave room for ingeniously applying them to prove equivalences with less effort than the truth table method. Theorem 3.4.5 then adds the reassurance that the axioms are guaranteed to prove every equivalence, which is a great punchline for this section. But we don't want to mislead you: it's important to realize that using the strategy we gave for applying the axioms involves essentially the same effort it would take to construct truth tables, and there is no guarantee that applying the axioms will generally be any easier than using truth tables.

3.5 The SAT Problem

= whether a proposition is satisfiable
↳ can evaluate to T.

Determining whether or not a more complicated proposition is satisfiable is not so easy. How about this one?

$$(P \text{ OR } Q \text{ OR } R) \text{ AND } (\overline{P} \text{ OR } \overline{Q}) \text{ AND } (\overline{P} \text{ OR } \overline{R}) \text{ AND } (\overline{R} \text{ OR } \overline{Q})$$

The general problem of deciding whether a proposition is satisfiable is called *SAT*. One approach to SAT is to construct a truth table and check whether or not a **T** ever appears, but as with testing validity, this approach quickly bogs down for formulas with many variables because truth tables grow exponentially with the number of variables.

Is there a more efficient solution to SAT? In particular, is there some brilliant procedure that determines SAT in a number of steps that grows *polynomially*—like

↳ instead of polynomially
2^n
for T tables...

n^2 or n^{14}—instead of *exponentially*—2^n—whether any given proposition of size n is satisfiable or not? No one knows. And an awful lot hangs on the answer.

The general definition of an "efficient" procedure is one that runs in *polynomial time*, that is, that runs in a number of basic steps bounded by a polynomial in s, where s is the size of an input. It turns out that an efficient solution to SAT would immediately imply efficient solutions to many other important problems involving scheduling, routing, resource allocation, and circuit verification across multiple disciplines including programming, algebra, finance, and political theory. This would be wonderful, but there would also be worldwide chaos. Decrypting coded messages would also become an easy task, so online financial transactions would be insecure and secret communications could be read by everyone. Why this would happen is explained in Section 8.12.

Of course, the situation is the same for validity checking, since you can check for validity by checking for satisfiability of a negated formula. This also explains why the simplification of formulas mentioned in Section 3.2 would be hard—validity testing is a special case of determining if a formula simplifies to **T**.

Recently there has been exciting progress on *SAT-solvers* for practical applications like digital circuit verification. These programs find satisfying assignments with amazing efficiency even for formulas with millions of variables. Unfortunately, it's hard to predict which kind of formulas are amenable to SAT-solver methods, and for formulas that are *un*satisfiable, SAT-solvers generally get nowhere.

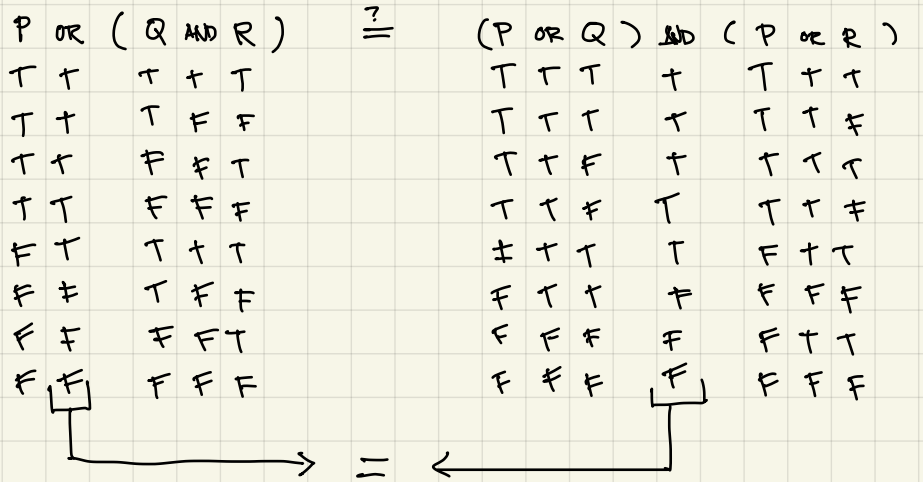
So no one has a good idea how to solve SAT in polynomial time, or how to prove that it can't be done—researchers are completely stuck. The problem of determining whether or not SAT has a polynomial time solution is known as the "**P** vs. **NP**" problem.[1] It is the outstanding unanswered question in theoretical computer science. It is also one of the seven Millenium Problems: the Clay Institute will award you $1,000,000 if you solve the **P** vs. **NP** problem.

[1] **P** stands for problems whose instances can be solved in time that grows polynomially with the size of the instance. **NP** stands for ***n****ondeterministtic* ***p****olynomial time*, but we'll leave an explanation of what that is to texts on the theory of computational complexity.

Sesh 4: May 3, 2024

Problem 1.
Prove by truth table that OR distributes over AND:

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



Problem 2.
This problem[1] examines whether the following specifications are *satisfiable*:

↳ can evaluate to T.

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 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.

(a) 1.a : $\overline{L} \to Q$
1.b : $\overline{L} \to B$
1.c : $\overline{L} \leftrightarrow N$
2 : $\overline{Q} \to B$
3 : $\overline{B}$

(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.

(b) Let $L=T$, $Q=T$, $B=F$, $N=F$
($L=T$: to make implication have ≠ hypo.; $Q=T$: ; $B=F$: make (3) true; $N=F$: so that $N \to \overline{L}$ is true.)

Then,

1.a = T	2. = T
1.b = T	3. = T
1.c = T	

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

(c.) $\bar{L} \to Q = L$ or Q

$\bar{L} \to B = L$ or B

$\bar{L} \leftrightarrow N = (L \text{ or } N)$ and $(\bar{N} \text{ or } \bar{L})$

$\bar{Q} \to B = Q$ or B

$\bar{B} = \bar{B}$

AND everything

$= (L+Q)(L+B)(L+N)(\bar{N}+\bar{L})(Q+B)(\bar{B})$

$= (L + QBN)(\bar{N}+\bar{L})(Q+B)\bar{B}$

$= (\bar{B}L + QB\bar{N}\bar{B})(\bar{N}+\bar{L})(Q+B)$ ($\bar{N}$ is F)

$= \bar{B}L(\bar{N}+\bar{L})(Q+B)$

$= (\bar{B}L\bar{N})(Q+B)$

$= \bar{B}L\bar{N}Q + \bar{B}L\bar{N}\bar{B}$

$= \bar{B}L\bar{N}Q$

→ true only for $L=T, N=F, Q=T, B=T$

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

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

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 "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.

↳ the second statement ("common") has an implicit declaration that T will only be possible after H. i.e. the future of H and $\bar{H}$ is declared.

↳ On the other hand, the first MATHEMATICAL statement does not tell us about the case where C = true. It only tells us about the case of C = false, nothing more.

Problem 4.
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, o_1, o_0. The 2-bit word a_1a_0 gives the binary representation of an integer, s between 0 and 3. The 3-bit word co_1o_0 gives the binary representation of $s + b$. The output c is called the *final carry bit*.

So if s and b were both 1, then the value of a_1a_0 would be `01` and the value of the output co_1o_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 $s = 3$ and $b = 1$. In that case, the value of co_1o_0 is `100`, namely, the binary representation of $3 + 1$.

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

$$
\begin{aligned}
c_0 &= b \\
o_0 &= a_0 \text{ XOR } c_0 \\
c_1 &= a_0 \text{ AND } c_0 && \text{the carry into column 1} \\
o_1 &= a_1 \text{ XOR } c_1 \\
c_2 &= a_1 \text{ AND } c_1 && \text{the carry into column 2} \\
c &= c_2.
\end{aligned}
$$

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

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

(b) Write similar definitions for the digits and carries in the sum of two $n+1$-bit binary numbers $a_n \ldots a_1a_0$ and $b_n \ldots b_1b_0$.

$c_0 = 0$
$o_i = c_i \text{ xor } (b_i \text{ xor } a_i)$ for $0 \leq i \leq n$
$c_{i+1} = a_ib_i + a_ic_i + b_ic_i$ for $0 \leq i \leq n$
$c = c_{n+1}$

c_i	0	0	0	0	1	1	1
b_i	1	1	0	0	1	1	0
a_i	0	1	0	1	0	1	0
o_i	1	0	0	1	0	1	0
b_i xor a_i	1	0	0	1	1	0	1
c_i xor (b_i xor a_i)	1	0	0	1	0	1	0

equiv

c_i	0	0	0	0	1	1	1
b_i	1	1	0	0	1	1	0
a_i	0	1	0	1	0	1	0
c_{i+1}	0	1	0	0	1	1	0
b_i AND a_i	0	1	0	0	0	1	0

by observation ...

$ab + ac + bc$

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

Solution. 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.[2]