Mathematics for Computer Science

revised Monday 18th May, 2015, 01:43

Eric Lehman
Google Inc.

F Thomson Leighton
Department of Mathematics
and the Computer Science and AI Laboratory,
Massachussetts Institute of Technology;
Akamai Technologies

Albert R Meyer
Department of Electrical Engineering and Computer Science
and the Computer Science and AI Laboratory,
Massachussetts Institute of Technology

Contents

3 Logical Formulas

It is amazing that people manage to cope with all the ambiguities in the English language. Here are some sentences that illustrate the issue:

- "You may have cake, or you may have ice cream."
- "If pigs can fly, then you can understand the Chebyshev bound."
- "If you can solve any problem we come up with, then you get an *A* for the course."
- "Every American has a dream."

What *precisely* do these sentences mean? Can you have both cake and ice cream or must you choose just one dessert? Pigs can't fly, so does the second sentence say anything about your understanding the Chebyshev bound? If you can solve some problems we come up with, can you get an *A* for the course? And if you can't solve a single one of the problems, does it mean you can't get an *A*? Finally, does the last sentence imply that all Americans have the same dream—say of owning a house—or might different Americans have different dreams—say, Eric dreams of designing a killer software application, Tom of being a tennis champion, Albert of being able to sing?

Some uncertainty is tolerable in normal conversation. But when we need to formulate ideas precisely—as in mathematics and programming—the ambiguities inherent in everyday language can be a real problem. We can't hope to make an exact argument if we're not sure exactly what the statements mean. So before we start into mathematics, we need to investigate the problem of how to talk about mathematics.

To get around the ambiguity of English, mathematicians have devised a special language for talking about logical relationships. This language mostly uses ordinary English words and phrases such as "or," "implies," and "for all." But mathematicians give these words precise and unambiguous definitions.

Surprisingly, in the midst of learning the language of logic, we'll come across the most important open problem in computer science—a problem whose solution could change the world.

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:

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 ($\text{NOT}(A)$ 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 ($\text{NOT}(A)$ 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 ($\text{NOT}(A)$ 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 ($\text{NOT}(A)$ 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

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

the *contrapositive* of the implication "P IMPLIES Q." The truth table shows that an implication and its contrapositive are equivalent—they are just different ways of saying the same thing.

In contrast, the *converse* of "P IMPLIES Q" is the statement "Q IMPLIES P." The converse to our example is:

If I am grumpy, then I am hungry.

This sounds like a rather different contention, and a truth table confirms this suspicion:

P	Q	P IMPLIES Q	Q IMPLIES P
T	**T**	**T**	**T**
T	**F**	**F**	**T**
F	**T**	**T**	**F**
F	**F**	**T**	**T**

Now the highlighted columns differ in the second and third row, confirming that an implication is generally *not* equivalent to its converse.

One final relationship: an implication and its converse together are equivalent to an iff statement, specifically, to these two statements together. For example,

If I am grumpy then I am hungry, and if I am hungry then I am grumpy.

are equivalent to the single statement:

I am grumpy iff I am hungry.

Once again, we can verify this with a truth table.

P	Q	(P IMPLIES Q)	AND	(Q IMPLIES P)	P IFF Q
T	**T**	**T**	**T**	**T**	**T**
T	**F**	**F**	**F**	**T**	**F**
F	**T**	**T**	**F**	**F**	**F**
F	**F**	**T**	**T**	**T**	**T**

The fourth column giving the truth values of

(P IMPLIES Q) AND (Q IMPLIES P)

is the same as the sixth column giving the truth values of P IFF Q, which confirms that the AND of the implications is equivalent to the IFF statement.

3.3.2 Validity and Satisfiability

A *valid* formula is one which is *always* true, no matter what truth values its variables may have. The simplest example is

$$P \text{ OR NOT}(P).$$

You can think about valid formulas as capturing fundamental logical truths. For example, a property of implication that we take for granted is that if one statement implies a second one, and the second one implies a third, then the first implies the third. The following valid formula confirms the truth of this property of implication.

$$[(P \text{ IMPLIES } Q) \text{ AND } (Q \text{ IMPLIES } R)] \text{ IMPLIES } (P \text{ IMPLIES } R).$$

Equivalence of formulas is really a special case of validity. Namely, statements F and G are equivalent precisely when the statement $(F \text{ IFF } G)$ is valid. For example, the equivalence of the expressions (3.3) and (3.2) means that

$$(A \text{ OR } B) \text{ IFF } (A \text{ OR } (\text{NOT}(A) \text{ AND } B))$$

is valid. Of course, validity can also be viewed as an aspect of equivalence. Namely, a formula is valid iff it is equivalent to **T**.

A *satisfiable* formula is one which can *sometimes* be true—that is, there is some assignment of truth values to its variables that makes it true. One way satisfiability comes up is when there are a collection of system specifications. The job of the system designer is to come up with a system that follows all the specs. This means that the AND of all the specs must be satisfiable or the designer's job will be impossible (see Problem 3.12).

There is also a close relationship between validity and satisfiability: a statement P is satisfiable iff its negation $\text{NOT}(P)$ is *not* valid.

3.4 The Algebra of Propositions

3.4.1 Propositions in Normal Form

Every propositional formula is equivalent to a "sum-of-products" or *disjunctive form*. More precisely, a disjunctive form is simply an OR of AND-terms, where each AND-term is an AND of variables or negations of variables, for example,

$$(A \text{ AND } B) \text{ OR } (A \text{ AND } C). \tag{3.4}$$

You can read a disjunctive form for any propositional formula directly from its truth table. For example, the formula

$$A \text{ AND } (B \text{ OR } C) \tag{3.5}$$

has truth table:

A	B	C	A AND (B OR C)
T	**T**	**T**	**T**
T	**T**	**F**	**T**
T	**F**	**T**	**T**
T	**F**	**F**	**F**
F	**T**	**T**	**F**
F	**T**	**F**	**F**
F	**F**	**T**	**F**
F	**F**	**F**	**F**

The formula (3.5) is true in the first row when A, B, and C are all true, that is, where A AND B AND C is true. It is also true in the second row where A AND B AND $\overline{C}$ is true, and in the third row when A AND $\overline{B}$ AND C is true, and that's all. So (3.5) is true exactly when

$$(A \text{ AND } B \text{ AND } C) \text{ OR } (A \text{ AND } B \text{ AND } \overline{C}) \text{ OR } (A \text{ AND } \overline{B} \text{ AND } C) \tag{3.6}$$

is true.

Theorem 3.4.1. *[Distributive Law of* AND *over* OR*]*

$$A \text{ AND } (B \text{ OR } C) \textit{ is equivalent to } (A \text{ AND } B) \text{ OR } (A \text{ AND } C).$$

Theorem 3.4.1 is called a *distributive law* because of its resemblance to the distributivity of products over sums in arithmetic.

Similarly, we have (Problem 3.10):

Theorem 3.4.2. *[Distributive Law of* OR *over* AND*]*

$$A \text{ OR } (B \text{ AND } C) \textit{ is equivalent to } (A \text{ OR } B) \text{ AND } (A \text{ OR } C).$$

Note the contrast between Theorem 3.4.2 and arithmetic, where sums do not distribute over products.

The expression (3.6) is a disjunctive form where each AND-term is an AND of *every one* of the variables or their negations in turn. An expression of this form is called a *disjunctive normal form* (*DNF*). A DNF formula can often be simplified into a smaller disjunctive form. For example, the DNF (3.6) further simplifies to the equivalent disjunctive form (3.4) above.

Applying the same reasoning to the **F** entries of a truth table yields a *conjunctive form* for any formula—an AND of OR-terms in which the OR-terms are OR's only of variables or their negations. For example, formula (3.5) is false in the fourth row of its truth table (3.4.1) where A is **T**, B is **F** and C is **F**. But this is exactly the one row where $(\overline{A}$ OR B OR $C)$ is **F**! Likewise, the (3.5) is false in the fifth row which is exactly where $(A$ OR $\overline{B}$ OR $\overline{C})$ is **F**. This means that (3.5) will be **F** whenever the AND of these two OR-terms is false. Continuing in this way with the OR-terms corresponding to the remaining three rows where (3.5) is false, we get a *conjunctive normal form* (*CNF*) that is equivalent to (3.5), namely,

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

The methods above can be applied to any truth table, which implies

Theorem 3.4.3. *Every propositional formula is equivalent to both a disjunctive normal form and a conjunctive normal form.*

3.4.2 Proving Equivalences

A check of equivalence or validity by truth table runs out of steam pretty quickly: a proposition with n variables has a truth table with 2^n lines, so the effort required to check a proposition grows exponentially with the number of variables. For a proposition with just 30 variables, that's already over a billion lines to check!

An alternative approach that *sometimes* helps is to use algebra to prove equivalence. A lot of different operators may appear in a propositional formula, so a useful first step is to get rid of all but three: AND, OR, and NOT. This is easy because each of the operators is equivalent to a simple formula using only these three. For example, A IMPLIES B is equivalent to NOT(A) OR B. Formulas using only AND, OR, and NOT for the remaining operators are left to Problem 3.13.

We list below a bunch of equivalence axioms with the symbol " $\longleftrightarrow$ " between equivalent formulas. These axioms are important because they are all that's needed to prove every possible equivalence. We'll start with some equivalences for AND's that look like the familiar ones for multiplication of numbers:

$$A \text{ AND } B \longleftrightarrow B \text{ AND } A \qquad \text{(commutativity of AND)} \tag{3.7}$$
$$(A \text{ AND } B) \text{ AND } C \longleftrightarrow A \text{ AND } (B \text{ AND } C) \qquad \text{(associativity of AND)} \tag{3.8}$$
$$\mathbf{T} \text{ AND } A \longleftrightarrow A \qquad \text{(identity for AND)}$$
$$\mathbf{F} \text{ AND } A \longleftrightarrow \mathbf{F} \qquad \text{(zero for AND)}$$

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

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

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.

3.6 Predicate Formulas

3.6.1 Quantifiers

The "for all" notation, $\forall$, has already made an early appearance in Section 1.1. For example, the predicate

$$"x^2 \geq 0"$$

[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.

is always true when x is a real number. That is,

$$\forall x \in \mathbb{R}.\, x^2 \geq 0$$

is a true statement. On the other hand, the predicate

$$\text{“}5x^2 - 7 = 0\text{”}$$

is only sometimes true; specifically, when $x = \pm\sqrt{7/5}$. There is a "there exists" notation, $\exists$, to indicate that a predicate is true for at least one, but not necessarily all objects. So

$$\exists x \in \mathbb{R}.\, 5x^2 - 7 = 0$$

is true, while

$$\forall x \in \mathbb{R}.\, 5x^2 - 7 = 0$$

is not true.

There are several ways to express the notions of "always true" and "sometimes true" in English. The table below gives some general formats on the left and specific examples using those formats on the right. You can expect to see such phrases hundreds of times in mathematical writing!

Always True

For all $x \in D$, $P(x)$ is true.	For all $x \in \mathbb{R}$, $x^2 \geq 0$.
$P(x)$ is true for every x in the set, D.	$x^2 \geq 0$ for every $x \in \mathbb{R}$.

Sometimes True

There is an $x \in D$ such that $P(x)$ is true.	There is an $x \in \mathbb{R}$ such that $5x^2 - 7 = 0$.
$P(x)$ is true for some x in the set, D.	$5x^2 - 7 = 0$ for some $x \in \mathbb{R}$.
$P(x)$ is true for at least one $x \in D$.	$5x^2 - 7 = 0$ for at least one $x \in \mathbb{R}$.

All these sentences "quantify" how often the predicate is true. Specifically, an assertion that a predicate is always true is called a *universal quantification*, and an assertion that a predicate is sometimes true is an *existential quantification*. Sometimes the English sentences are unclear with respect to quantification:

$$\begin{aligned}&\text{If you can solve any problem we come up with,}\\ &\qquad\text{then you get an } A \text{ for the course.}\end{aligned} \tag{3.16}$$

The phrase "you can solve any problem we can come up with" could reasonably be interpreted as either a universal or existential quantification:

$$\text{you can solve } \textit{every} \text{ problem we come up with,} \tag{3.17}$$

or maybe

you can solve *at least one* problem we come up with. (3.18)

To be precise, let Probs be the set of problems we come up with, Solves(x) be the predicate "You can solve problem x," and G be the proposition, "You get an A for the course." Then the two different interpretations of (3.16) can be written as follows:

$$(\forall x \in \text{Probs}.\, \text{Solves}(x)) \text{ IMPLIES } G, \qquad \text{for (3.17)},$$
$$(\exists x \in \text{Probs}.\, \text{Solves}(x)) \text{ IMPLIES } G. \qquad \text{for (3.18)}.$$

3.6.2 Mixing Quantifiers

Many mathematical statements involve several quantifiers. For example, we already described

> Goldbach's Conjecture 1.1.8: Every even integer greater than 2 is the sum of two primes.

Let's write this out in more detail to be precise about the quantification:

> For every even integer n greater than 2, there exist primes p and q such that $n = p + q$.

Let Evens be the set of even integers greater than 2, and let Primes be the set of primes. Then we can write Goldbach's Conjecture in logic notation as follows:

$$\underbrace{\forall n \in \text{Evens}}_{\substack{\text{for every even} \\ \text{integer } n > 2}} \underbrace{\exists p \in \text{Primes}\, \exists q \in \text{Primes}.}_{\substack{\text{there exist primes} \\ p \text{ and } q \text{ such that}}} n = p + q.$$

3.6.3 Order of Quantifiers

Swapping the order of different kinds of quantifiers (existential or universal) usually changes the meaning of a proposition. For example, let's return to one of our initial, confusing statements:

"Every American has a dream."

This sentence is ambiguous because the order of quantifiers is unclear. Let A be the set of Americans, let D be the set of dreams, and define the predicate $H(a, d)$ to be "American a has dream d." Now the sentence could mean there is a single dream that every American shares—such as the dream of owning their own home:

$$\exists d \in D\, \forall a \in A.\, H(a, d)$$

Or it could mean that every American has a personal dream:

$$\forall a \in A\, \exists d \in D.\, H(a, d)$$

For example, some Americans may dream of a peaceful retirement, while others dream of continuing practicing their profession as long as they live, and still others may dream of being so rich they needn't think about work at all.

Swapping quantifiers in Goldbach's Conjecture creates a patently false statement that every even number ≥ 2 is the sum of *the same* two primes:

$$\underbrace{\exists\, p \in \text{Primes}\, \exists\, q \in \text{Primes}.}_{\substack{\text{there exist primes}\\ p \text{ and } q \text{ such that}}} \underbrace{\forall n \in \text{Evens}}_{\substack{\text{for every even}\\ \text{integer } n > 2}} n = p + q.$$

3.6.4 Variables Over One Domain

When all the variables in a formula are understood to take values from the same nonempty set, D, it's conventional to omit mention of D. For example, instead of $\forall x \in D\, \exists y \in D.\, Q(x, y)$ we'd write $\forall x \exists y.\, Q(x, y)$. The unnamed nonempty set that x and y range over is called the *domain of discourse*, or just plain *domain*, of the formula.

It's easy to arrange for all the variables to range over one domain. For example, Goldbach's Conjecture could be expressed with all variables ranging over the domain $\mathbb{N}$ as

$$\forall n.\, n \in \text{Evens IMPLIES } (\exists\, p\, \exists q.\, p \in \text{Primes AND } q \in \text{Primes AND } n = p + q).$$

3.6.5 Negating Quantifiers

There is a simple relationship between the two kinds of quantifiers. The following two sentences mean the same thing:

> Not everyone likes ice cream.
>
> There is someone who does not like ice cream.

The equivalence of these sentences is a instance of a general equivalence that holds between predicate formulas:

$$\text{NOT}(\forall x.\, P(x)) \quad \text{is equivalent to} \quad \exists x.\, \text{NOT}(P(x)). \tag{3.19}$$

Similarly, these sentences mean the same thing:

> There is no one who likes being mocked.
>
> Everyone dislikes being mocked.

The corresponding predicate formula equivalence is

$$\text{NOT}(\exists x.\, P(x)) \quad \text{is equivalent to} \quad \forall x.\, \text{NOT}(P(x)). \tag{3.20}$$

The general principle is that *moving a* NOT *across a quantifier changes the kind of quantifier.* Note that (3.20) follows from negating both sides of (3.19).

3.6.6 Validity for Predicate Formulas

The idea of validity extends to predicate formulas, but to be valid, a formula now must evaluate to true no matter what the domain of discourse may be, no matter what values its variables may take over the domain, and no matter what interpretations its predicate variables may be given. For example, the equivalence (3.19) that gives the rule for negating a universal quantifier means that the following formula is valid:

$$\text{NOT}(\forall x.\, P(x)) \text{ IFF } \exists x.\, \text{NOT}(P(x)). \tag{3.21}$$

Another useful example of a valid assertion is

$$\exists x \forall y.\, P(x, y) \text{ IMPLIES } \forall y \exists x.\, P(x, y). \tag{3.22}$$

Here's an explanation why this is valid:

> Let D be the domain for the variables and P_0 be some binary predicate[2] on D. We need to show that if
>
> $$\exists x \in D.\, \forall y \in D.\, P_0(x, y) \tag{3.23}$$
>
> holds under this interpretation, then so does
>
> $$\forall y \in D\, \exists x \in D.\, P_0(x, y). \tag{3.24}$$
>
> So suppose (3.23) is true. Then by definition of $\exists$, this means that some element $d_0 \in D$ has the property that
>
> $$\forall y \in D.\, P_0(d_0, y).$$
>
> By definition of $\forall$, this means that
>
> $$P_0(d_0, d)$$
>
> is true for all $d \in D$. So given any $d \in D$, there is an element in D, namely, d_0, such that $P_0(d_0, d)$ is true. But that's exactly what (3.24) means, so we've proved that (3.24) holds under this interpretation, as required.

[2] That is, a predicate that depends on two variables.

We hope this is helpful as an explanation, but we don't really want to call it a "proof." The problem is that with something as basic as (3.22), it's hard to see what more elementary axioms are ok to use in proving it. What the explanation above did was translate the logical formula (3.22) into English and then appeal to the meaning, in English, of "for all" and "there exists" as justification.

In contrast to (3.22), the formula

$$\forall y \exists x.\ P(x, y) \ \text{IMPLIES}\ \exists x \forall y.\ P(x, y). \tag{3.25}$$

is *not* valid. We can prove this just by describing an interpretation where the hypothesis, $\forall y \exists x.\ P(x, y)$, is true but the conclusion, $\exists x \forall y.\ P(x, y)$, is not true. For example, let the domain be the integers and $P(x, y)$ mean $x > y$. Then the hypothesis would be true because, given a value, n, for y we could choose the value of x to be $n + 1$, for example. But under this interpretation the conclusion asserts that there is an integer that is bigger than all integers, which is certainly false. An interpretation like this that falsifies an assertion is called a *counter model* to that assertion.

3.7 References

[18]

Problems for Section 3.1

Practice Problems

Problem 3.1.
Some people are uncomfortable with the idea that from a false hypothesis you can prove everything, and instead of having P IMPLIES Q be true when P is false, they want P IMPLIES Q to be false when P is false. This would lead to IMPLIES having the same truth table as what propositional connective?

Problem 3.2.
Your class has a textbook and a final exam. Let P, Q, and R be the following propositions:

$P ::=$ You get an A on the final exam.

Q::= You do every exercise in the book.

R::= You get an A in the class.

Translate following assertions into propositional formulas using P, Q, R and the propositional connectives AND, NOT, IMPLIES.

(a) You get an A in the class, but you do not do every exercise in the book.

(b) You get an A on the final, you do every exercise in the book, and you get an A in the class.

(c) To get an A in the class, it is necessary for you to get an A on the final.

(d) You get an A on the final, but you don't do every exercise in this book; nevertheless, you get an A in this class.

Class Problems

Problem 3.3.

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 "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.

Homework Problems

Problem 3.4.
Describe a simple procedure which, given a positive integer argument, n, produces a width n array of truth-values whose rows would be all the possible truth-value assignments for n propositional variables. For example, for $n = 2$, the array would be:

T	**T**
T	**F**
F	**T**
F	**F**

Your description can be in English, or a simple program in some familiar language such as Python or Java. If you do write a program, be sure to include some sample output.

Problems for Section 3.2

Class Problems

Problem 3.5.
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 $a_n, \ldots, a_1, a_0$ 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 $c = c_{n+1}$.

(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 $b_n \ldots b_1 b_0$.

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?

Homework Problems

Problem 3.6.

There are adder circuits that are *much* faster, and only slightly larger, than the ripple-carry circuits of Problem 3.5. They work by computing the values in later columns for both a carry of 0 and a carry of 1, *in parallel*. Then, when the carry from the earlier columns finally arrives, the pre-computed answer can be quickly selected. We'll illustrate this idea by working out the equations for an $(n+1)$-bit parallel half-adder.

Parallel half-adders are built out of parallel *add1* modules. An $(n+1)$-bit *add1* module takes as input the $(n+1)$-bit binary representation, $a_n \ldots a_1 a_0$, of an integer, s, and produces as output the binary representation, $c\ p_n \ldots p_1 p_0$, of $s+1$.

(a) A 1-bit *add1* module just has input a_0. Write propositional formulas for its outputs c and p_0.

(b) Explain how to build an $(n+1)$-bit parallel half-adder from an $(n+1)$-bit *add1* module by writing a propositional formula for the half-adder output, o_i, using only the variables a_i, p_i, and b.

We can build a double-size *add1* module with $2(n+1)$ inputs using two single-size *add1* modules with $n+1$ inputs. Suppose the inputs of the double-size module are $a_{2n+1}, \ldots, a_1, a_0$ and the outputs are c, $p_{2n+1}, \ldots, p_1, p_0$. The setup is illustrated in Figure 3.1.

Namely, the first single size *add1* module handles the first $n+1$ inputs. The inputs to this module are the low-order $n+1$ input bits $a_n, \ldots, a_1, a_0$, and its outputs will serve as the first $n+1$ outputs $p_n, \ldots, p_1, p_0$ of the double-size module. Let $c_{(1)}$ be the remaining carry output from this module.

The inputs to the second single-size module are the higher-order $n+1$ input bits $a_{2n+1}, \ldots, a_{n+2}, a_{n+1}$. Call its first $n+1$ outputs $r_n, \ldots, r_1, r_0$ and let $c_{(2)}$ be its carry.

(c) Write a formula for the carry, c, in terms of $c_{(1)}$ and $c_{(2)}$.

(d) Complete the specification of the double-size module by writing propositional formulas for the remaining outputs, p_i, for $n+1 \leq i \leq 2n+1$. The formula for p_i should only involve the variables a_i, $r_{i-(n+1)}$, and $c_{(1)}$.

(e) Parallel half-adders are exponentially faster than ripple-carry half-adders. Confirm this by determining the largest number of propositional operations required to compute any one output bit of an n-bit add module. (You may assume n is a power of 2.)

Exam Problems

Problem 3.7.
There are exactly two truth environments (assignments) for the variables M, N, P, Q, R, S that satisfy the following formula:

$$\underbrace{(\overline{P} \text{ OR } Q)}_{\text{clause (1)}} \text{ AND } \underbrace{(\overline{Q} \text{ OR } R)}_{\text{clause (2)}} \text{ AND } \underbrace{(\overline{R} \text{ OR } S)}_{\text{clause (3)}} \text{ AND } \underbrace{(\overline{S} \text{ OR } P)}_{\text{clause (4)}} \text{ AND } M \text{ AND } \overline{N}$$

(a) This claim could be proved by truth-table. How many rows would the truth table have?

(b) Instead of a truth-table, prove this claim with an argument by cases according to the truth value of P.

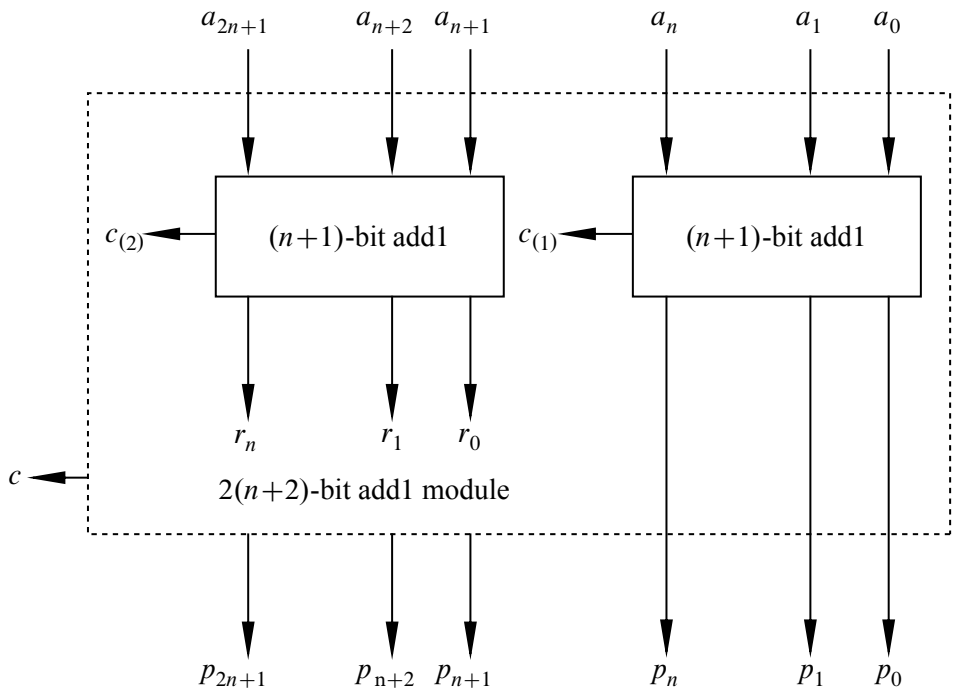

Figure 3.1 Structure of a Double-size *add1* Module.

Problems for Section 3.3

Practice Problems

Problem 3.8.
Indicate whether each of the following propositional formulas is valid (V), satisfiable but not valid (S), or not satisfiable (N). For the satisfiable ones, indicate a satisfying truth assignment.

$$
\begin{aligned}
&M \text{ IMPLIES } Q\\
&M \text{ IMPLIES } (\overline{P} \text{ OR } \overline{Q})\\
&M \text{ IMPLIES } [M \text{ AND } (P \text{ IMPLIES } M)]\\
(P \text{ OR } Q) &\text{ IMPLIES } Q\\
(P \text{ OR } Q) &\text{ IMPLIES } (\overline{P} \text{ AND } \overline{Q})\\
(P \text{ OR } Q) &\text{ IMPLIES } [M \text{ AND } (P \text{ IMPLIES } M)]\\
(P \text{ XOR } Q) &\text{ IMPLIES } Q\\
(P \text{ XOR } Q) &\text{ IMPLIES } (\overline{P} \text{ OR } \overline{Q})\\
(P \text{ XOR } Q) &\text{ IMPLIES } [M \text{ AND } (P \text{ IMPLIES } M)]
\end{aligned}
$$

Problem 3.9.
Prove that the propositional formulas

$$P \text{ OR } Q \text{ OR } R$$

and

$$(P \text{ AND NOT}(Q)) \text{ OR } (Q \text{ AND NOT}(R)) \text{ OR } (R \text{ AND NOT}(P)) \text{ OR } (P \text{ AND } Q \text{ AND } R).$$

are equivalent.

Problem 3.10.
Prove by truth table that OR distributes over AND, namely,

$$P \text{ OR } (Q \text{ AND } R) \quad \text{is equivalent to} \quad (P \text{ OR } Q) \text{ AND } (P \text{ OR } R) \tag{3.26}$$

Class Problems

Problem 3.11. **(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*—which makes it true. Explain why

P is valid iff NOT(P) is *not* satisfiable.

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

Problem 3.12.
This problem[3] 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 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.

[3] Revised from Rosen, 5th edition, Exercise 1.1.36

(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.

Problems for Section 3.4

Practice Problems

Problem 3.13.
A half dozen different operators may appear in propositional formulas, but just AND, OR, and NOT are enough to do the job. That is because each of the operators is equivalent to a simple formula using only these three operators. For example, A IMPLIES B is equivalent to NOT(A) OR B. So all occurences of IMPLIES in a formula can be replaced using just NOT and OR.

(a) Write formulas using only AND, OR, NOT that are equivalent to each of A IFF B and A XOR B. Conclude that every propositional formula is equivalent to an AND-OR-NOT formula.

(b) Explain why you don't even need AND.

(c) Explain how to get by with the single operator NAND where A NAND B is equivalent by definition to NOT(A AND B).

Class Problems

Problem 3.14.
The propositional connective NOR is defined by the rule

$$P \text{ NOR } Q \ ::= \ (\text{NOT}(P) \text{ AND NOT}(Q)).$$

Explain why every propositional formula—possibly involving any of the usual operators such as IMPLIES, XOR, ...—is equivalent to one whose only connective is NOR.

Problem 3.15.
Explain how to find a conjunctive form for a propositional formula directly from a disjunctive form for its complement.

Homework Problems

Problem 3.16.
Use the equivalence axioms of Section 3.4.2 to convert the following formula to disjunctive form:

$$A \text{ XOR } B \text{ XOR } C.$$

Problems for Section 3.5

Homework Problems

Problem 3.17.
A 3-conjunctive form (3CF) formula is a conjunctive form formula in which each OR-term is an OR of at most 3 variables or negations of variables. Although it may be hard to tell if a propositional formula, F, is satisfiable, it is always easy to construct a formula, $\mathcal{C}(F)$, that is

- in 3-conjunctive form,
- has at most 24 times as many occurrences of variables as F, and
- is satisfiable iff F is satisfiable.

To construct $\mathcal{C}(F)$, introduce a different new variables for each operator that occurs in F. For example, if F was

$$((P \text{ XOR } Q) \text{ XOR } R) \text{ OR } (\overline{P} \text{ AND } S) \tag{3.27}$$

we might use new variables X_1, X_2, O, and A corresponding to the operator occurrences as follows:

$$((P \underbrace{\text{XOR}}_{X_1} Q) \underbrace{\text{XOR}}_{X_2} R) \underbrace{\text{OR}}_{O} (\overline{P} \underbrace{\text{AND}}_{A} S).$$

Next we write a formula that constrains each new variable to have the same truth value as the subformula determined by its corresponding operator. For the example above, these constraining formulas would be

$$\begin{aligned} X_1 &\text{ IFF } (P \text{ XOR } Q),\\ X_2 &\text{ IFF } (X_1 \text{ XOR } R),\\ A &\text{ IFF } (\overline{P} \text{ AND } S),\\ O &\text{ IFF } (X_2 \text{ OR } A) \end{aligned}$$

(a) Explain why the AND of the four constraining formulas above along with a fifth formula consisting of just the variable O will be satisfiable iff (3.27) is satisfiable.

(b) Explain why each constraining formula will be equivalent to a 3CF formula with at most 24 occurrences of variables.

(c) Using the ideas illustrated in the previous parts, explain how to construct $\mathcal{C}(F)$ for an arbitrary propositional formula, F.

Problem 3.18.
It doesn't matter whether we formulate the SAT problem (Section 3.5 in terms of propositional formulas or digital circuits. Here's why:

Let f be a Boolean function of k variables. That is, $f : \{\mathbf{T}, \mathbf{F}\}^k \to \{\mathbf{T}, \mathbf{F}\}$. When P is a propositional formula that has, among its variables, propositional variables labelled $X_1, \ldots, X_k$. For any truth values $b_1, \ldots, b_k \in \{\mathbf{T}, \mathbf{F}\}$, we let let $P(b_1, \ldots, b_k)$ be the result of substituting b_i for all occurrences of X_i in P, for $1 \leq i \leq k$.

If P_f is a formula such that $P_f(b_1, \ldots, b_k)$ is satisfiable exactly when $f(b_1, \ldots, b_k) = \mathbf{T}$, we'll say that P_f *SAT-represents* f.

Suppose there is a digital circuit using two-input, one-output binary gates (like the circuits for binary addition in Problems 3.5 and 3.6) that has n wires and computes the function f. Explain how to construct a formula P_f of size cn that SAT-represents f for some small constant c. (Letting $c = 6$ will work).

Conclude that the SAT problem for digital circuits—that is, determining if there is some set of input values that will lead a circuit to give output 1—is no more difficult than the SAT problem for propositional formulas.

Hint: Introduce a new variable for each wire. The idea is similar to the one used in Problem 3.17 to show that satisfiablity of 3CNF propositional formmulas is just as hard as for arbitrary formulas.

Problems for Section 3.6

Practice Problems

Problem 3.19.
For each of the following propositions:

1. $\forall x \, \exists y. \, 2x - y = 0$

2. $\forall x\, \exists y.\ x - 2y = 0$
3. $\forall x.\, x < 10$ IMPLIES $(\forall y.\ y < x$ IMPLIES $y < 9)$
4. $\forall x\, \exists y.\ [y > x \wedge \exists z.\ y + z = 100]$

determine which propositions are true when the variables range over:

(a) the nonnegative integers.

(b) the integers.

(c) the real numbers.

Problem 3.20.
Let $Q(x, y)$ be the statement

"x has been a contestant on television show y."

The universe of discourse for x is the set of all students at your school and for y is the set of all quiz shows that have ever been on television.

Determine whether or not each of the following expressions is logically equivalent to the sentence:

"No student at your school has ever been a contestant on a television quiz show."

(a) $\forall x\, \forall y.$ NOT$(Q(x, y))$

(b) $\exists x\, \exists y.$ NOT$(Q(x, y))$

(c) NOT$(\forall x\, \forall y.\ Q(x, y))$

(d) NOT$(\exists x\, \exists y.\ Q(x, y))$

Problem 3.21.
Find a counter model showing the following is not valid.

$$\exists x.P(x) \text{ IMPLIES } \forall x.P(x)$$

(Just define your counter model. You do not need to verify that it is correct.)

Problem 3.22.
Find a counter model showing the following is not valid.

$$[\exists x.\, P(x) \text{ AND } \exists x.Q(x)] \text{ IMPLIES } \exists x.[P(x) \text{ AND } Q(x)]$$

(Just define your counter model. You do not need to verify that it is correct.)

Problem 3.23.
Which of the following are *valid*?

(a) $\exists x \exists y.\ P(x, y)$ IMPLIES $\exists y \exists x.\ P(x, y)$

(b) $\forall x \exists y.\ Q(x, y)$ IMPLIES $\exists y \forall x.\ Q(x, y)$

(c) $\exists x \forall y.\ R(x, y)$ IMPLIES $\forall y \exists x.\ R(x, y)$

(d) NOT$(\exists x\ S(x))$ IFF $\forall x$ NOT$(S(x))$

Class Problems

Problem 3.24.
A media tycoon has an idea for an all-news television network called LNN: The Logic News Network. Each segment will begin with a definition of the domain of discourse and a few predicates. The day's happenings can then be communicated concisely in logic notation. For example, a broadcast might begin as follows:

> THIS IS LNN. The domain of discourse is
>
> {Albert, Ben, Claire, David, Emily}.
>
> Let $D(x)$ be a predicate that is true if x is deceitful. Let $L(x, y)$ be a predicate that is true if x likes y. Let $G(x, y)$ be a predicate that is true if x gave gifts to y.

Translate the following broadcasts in logic notation into (English) statements.

(a)

NOT$(D(\text{Ben})$ OR $D(\text{David}))$ IMPLIES $(L(\text{Albert}, \text{Ben})$ AND $L(\text{Ben}, \text{Albert}))$

(b)

$\forall x$ $((x = \text{Claire}$ AND NOT$(L(x, \text{Emily})))$ OR $(x \neq \text{Claire}$ AND $L(x, \text{Emily})))$ AND
$\forall x$ $((x = \text{David}$ AND $L(x, \text{Claire}))$ OR $(x \neq \text{David}$ AND NOT$(L(x, \text{Claire}))))$

(c)

$$\text{NOT}(D(\text{Claire})) \text{ IMPLIES } (G(\text{Albert}, \text{Ben}) \text{ AND } \exists x.\, G(\text{Ben}, x))$$

(d)

$$\forall x \exists y \exists z\ (y \neq z) \text{ AND } L(x, y) \text{ AND NOT}(L(x, z))$$

(e) How could you express "Everyone except for Claire likes Emily" using just propositional connectives *without* using any quantifiers ($\forall, \exists$)? Can you generalize to explain how *any* logical formula over this domain of discourse can be expressed without quantifiers? How big would the formula in the previous part be if it was expressed this way?

Problem 3.25.

The goal of this problem is to translate some assertions about binary strings into logic notation. The domain of discourse is the set of all finite-length binary strings: λ, 0, 1, 00, 01, 10, 11, 000, 001, ... (Here λ denotes the empty string.) In your translations, you may use all the ordinary logic symbols (including =), variables, and the binary symbols `0`, `1` denoting 0, 1.

A string like $\texttt{01}x\texttt{0}y$ of binary symbols and variables denotes the *concatenation* of the symbols and the binary strings represented by the variables. For example, if the value of x is `011` and the value of y is `1111`, then the value of $\texttt{01}x\texttt{0}y$ is the binary string `0101101111`.

Here are some examples of formulas and their English translations. Names for these predicates are listed in the third column so that you can reuse them in your solutions (as we do in the definition of the predicate NO-1S below).

Meaning	Formula	Name
x is a prefix of y	$\exists z\ (xz = y)$	$\text{PREFIX}(x, y)$
x is a substring of y	$\exists u \exists v\ (uxv = y)$	$\text{SUBSTRING}(x, y)$
x is empty or a string of 0's	$\text{NOT}(\text{SUBSTRING}(\texttt{1}, x))$	$\text{NO-1S}(x)$

(a) x consists of three copies of some string.

(b) x is an even-length string of 0's.

(c) x does not contain both a 0 and a 1.

(d) x is the binary representation of $2^k + 1$ for some integer $k \geq 0$.

(e) An elegant, slightly trickier way to define NO-1S(x) is:

$$\text{PREFIX}(x, 0x). \tag{*}$$

Explain why (*) is true only when x is a string of 0's.

Problem 3.26.
For each of the logical formulas, indicate whether or not it is true when the domain of discourse is $\mathbb{N}$, (the nonnegative integers 0, 1, 2, ...), $\mathbb{Z}$ (the integers), $\mathbb{Q}$ (the rationals), $\mathbb{R}$ (the real numbers), and $\mathbb{C}$ (the complex numbers). Add a brief explanation to the few cases that merit one.

$$\begin{aligned}
\exists x.\, x^2 &= 2 \\
\forall x. \exists y.\, x^2 &= y \\
\forall y. \exists x.\, x^2 &= y \\
\forall x \neq 0. \exists y.\, xy &= 1 \\
\exists x. \exists y.\, x + 2y = 2 \ &\text{ AND } \ 2x + 4y = 5
\end{aligned}$$

Problem 3.27.
Show that

$$(\forall x \exists y.\ P(x, y)) \longrightarrow \forall z.\ P(z, z)$$

is not valid by describing a counter-model.

Problem 3.28.
If the names of procedures or their parameters are used in separate places, it doesn't really matter if the same variable name happens to be appear, and it's always safe to change a "local" name to something brand new. The same thing happens in predicate formulas.

For example, we can rename the variable x in "$\forall x.P(x)$" to be "y" to obtain $\forall y.P(y)$ and these two formulas are equivalent. So a formula like

$$(\forall x.P(x)) \text{ AND } (\forall x.Q(x)) \tag{3.28}$$

can be rewritten as the equivalent formula

$$(\forall y.P(y)) \text{ AND } (\forall x.Q(x)), \tag{3.29}$$

which more clearly shows that the separate occurrences of $\forall x$ in (3.28) are unrelated.

Renaming variables in this way allows every predicate formula to be converted into an equivalent formula in which every variable name is used in only one way. Specifically, a predicate formula satisfies the *unique variable convention* if

- for every variable x, there is at most one quantified occurrence of x, that is, at most one occurrence of either "$\forall x$" or "$\exists x$," and moreover, "$\forall x$" and "$\exists x$" don't both occur, and
- if there is a subformula of the form $\forall x.F$ or the form $\exists x.F$, then all the occurrences of x that appear anywhere in the whole formula are within the formula F.

So formula (3.28) violates the unique variable convention because "$\forall x$" occurs twice, but formula (3.29) is OK.

A further example is the formula

$$[\forall x\, \exists y.\ P(x) \text{ AND } Q(x,y)] \text{ IMPLIES } (\exists x.\ R(x,z)) \text{ OR } \exists x\, \forall z.\ S(x,y,w,z). \tag{3.30}$$

Formula (3.30) violates the unique variable convention because there are three quantified occurrences of x in the formula, namely, the initial "$\forall x$" and then two occurrences of "$\exists x$" later. It violates the convention in others ways as well. For instance, there is an occurrence of y that is not inside the subformula $\exists y.\ P(x)$ AND $Q(y)$.

It turns out that *every* predicate formula can be changed into an equivalent formula that satisfies the unique variable convention—just by renaming some of the occurrences of its variables, as we did this when we renamed the first two occurrences of x in (3.28) into y's to obtain the equivalent formula (3.29).

(a) Rename occurrences of variables in (3.30) to obtain an equivalent formula that satisfies the unique variable convention. Try to rename as few occurrences as posible.

(b) Describe a general procedure for renaming variables in any predicate formula to obtain an equivalent formula satisfying the unique variable convention.

Homework Problems

Problem 3.29.

Express each of the following predicates and propositions in formal logic notation. The domain of discourse is the nonnegative integers, $\mathbb{N}$. Moreover, in addition to

the propositional operators, variables and quantifiers, you may define predicates using addition, multiplication, and equality symbols, and nonnegative integer *constants* (0, 1,...), but no *exponentiation* (like x^y). For example, the predicate "n is an even number" could be defined by either of the following formulas:

$$\exists m.\ (2m = n), \qquad \exists m.\ (m + m = n).$$

(a) m is a divisor of n.

(b) n is a prime number.

(c) n is a power of a prime.

Problem 3.30.
Translate the following sentence into a predicate formula:

> There is a student who has e-mailed at most two other people in the class, besides possibly himself.

The domain of discourse should be the set of students in the class; in addition, the only predicates that you may use are

- equality, and
- $E(x, y)$, meaning that "x has sent e-mail to y."

Problem 3.31.
Translate the following sentence into a predicate formula:

> There is a student who has emailed exactly two other people in the class, besides possibly herself.

The domain of discourse should be the set of students in the class; in addition, the only predicates that you may use are

- equality, and
- $E(x, y)$, meaning that "x has sent e-mail to y."

Exam Problems

Problem 3.32.

The following predicate logic formula is invalid:

$$\forall x, \exists y. P(x, y) \longrightarrow \exists y, \forall x. P(x, y)$$

Which of the following are counter models for it?

1. The predicate $P(x, y) =$ '$y \cdot x = 1$' where the domain of discourse is $\mathbb{Q}$.
2. The predicate $P(x, y) =$ '$y < x$' where the domain of discourse is $\mathbb{R}$.
3. The predicate $P(x, y) =$ '$y \cdot x = 2$' where the domain of discourse is $\mathbb{R}$ without 0.
4. The predicate $P(x, y) =$ '$yxy = x$' where the domain of discourse is the set of all binary strings, including the empty string.

Problem 3.33.

Some students from a large class will be lined up left to right. There will be at least two stduents in the line. Translate each of the following assertions into predicate formulas with the set of students in the class as the domain of discourse. The only predicates you may use are

- equality and,
- $F(x, y)$, meaning that "x is somewhere to the left of y in the line." For example, in the line "CDA", both $F(C, A)$ and $F(C, D)$ are true.

Once you have defined a formula for a predicate P you may use the abbreviation "P" in further formulas.

(a) Student x is in the line.

(b) Student x is first in line.

(c) Student x is immediately to the right of student y.

(d) Student x is second.

Problem 3.34.
We want to find predicate formulas about the nonnegative integers, $\mathbb{N}$, in which $\leq$ is the only predicate that appears, and no constants appear.

For example, there is such a formula defining the equality predicate:

$$[x = y] ::= [x \leq y \text{ AND } y \leq x].$$

Once predicate is shown to be expressible solely in terms of $\leq$, it may then be used in subsequent translations. For example,

$$[x > 0] ::= \exists y.\ \text{NOT}(x = y) \text{ AND } y \leq x.$$

(a) $[x = 0]$.

(b) $[x = y + 1]$

(c) $x = 3$