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 your account won't get hacked."
- "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 the security of your account? 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 which don't always match common usage.

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 If and Only If

Mathematicians commonly join propositions in an 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.

3.1.3 IMPLIES

The combining operation whose technical meaning is least intuitive 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.

We already mentioned that no one knows whether Goldbach's Conjecture, Proposition 1.1.6, is true or false. But that doesn't prevent us 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*!

Now let's figure out the truth of one of our original examples:

> If pigs fly, then your account won't get hacked.

Forget about pigs, we just need to figure out whether this proposition is true or false. 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*!

False Hypotheses

This mathematical convention—that an implication as a whole is considered true when its hypothesis is false—contrasts with common cases where implications are supposed to have some *causal* connection between their hypotheses and conclusions.

For example, we could agree—or at least hope—that the following statement is true:

> If you followed the security protocal, then your account won't get hacked.

We regard this implication as unproblematical because of the clear *causal* connection between security protocols and account hackability.

On the other hand, the statement:

> If pigs could fly, then your account won't get hacked,

would commonly be rejected as false—or at least silly—because porcine aeronautics have nothing to do with your account security. But mathematically, this implication counts as true.

It's important to accept the fact that mathematical implications ignore causal connections. This makes them a lot simpler than causal implications, but useful nevertheless. To illustrate this, suppose we have a system specification which consists of a series of, say, a dozen rules,[1]

If the system sensors are in condition 1,
then the system takes action 1.
If the system sensors are in condition 2,
then the system takes action 2.
⋮
If the system sensors are in condition 12,
then the system takes action 12.

Letting C_i be the proposition that the system sensors are in condition i, and A_i be the proposition that system takes action i, the specification can be restated more concisely by the logical formulas

$$C_1 \text{ IMPLIES } A_1,$$
$$C_2 \text{ IMPLIES } A_2,$$
$$\vdots$$
$$C_{12} \text{ IMPLIES } A_{12}.$$

Now the proposition that the system obeys the specification can be nicely expressed as a single logical formula by combining the formulas together with ANDs::

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

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. So in this case, the system is behaving

[1] Problem 3.17 concerns just such a system.

according to specification, and we accordingly want formula (3.1) to come out true. 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, all of whose hypotheses are false, to be true. This is exactly what the rule for mathematical implications accomplishes.

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 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 3.6 and 3.7). 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

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

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 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 3.17).

There is also a close relationship between validity and satisfiability: a statement P is satisfiable iff its negation 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 normal form* (DNF).

More precisely, a propositional variable A or its negation $bar(A)$ is called a *literal*, and an AND of literals involving *distinct* variables is called an AND-*clause* or AND-*of-literals*. For example,

$$A \text{ AND } \overline{B} \text{ AND } \overline{C}$$

is an AND-clause, but A AND B AND $\overline{B}$ AND C is not because B appears twice. Finally, a DNF is an OR of AND-clauses such as

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

You can read a DNF 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.

The expression (3.6) is a DNF where each AND-clause actually includes a literal for *every one* of the variables in the whole formula. We'll call such a formula a *full DNF*.

A DNF formula can often be simplified into a smaller DNF For example, the DNF (3.6) further simplifies to the equivalent DNF (3.4) above.

Applying the same reasoning to the **F** entries of a truth table yields a *conjunctive normal form* (CNF) for any formula—an AND of OR-*clauses*, where an OR-clause is an OR-*of-literals* from different variables.

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 the OR-clause ($\overline{A}$ OR B OR C) is **F**! Likewise, (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-clauses is false. Continuing in this way with the OR-clauses corresponding to the remaining three rows where (3.5) is false, we get a CNF that is equivalent to (3.5), namely,

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

Again, each OR-clause includes a literal for every one of the variables, that is, it is a *full* CNF

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

Theorem 3.4.1. *Every propositional formula is equivalent to both a full disjunctive normal form and a full 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 defining the remaining operators using only AND, OR and NOT are left to Problem 3.18.

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:

$$A \text{ AND } B \longleftrightarrow B \text{ AND } A \qquad \text{(commutativity of AND)} \tag{3.7}$$

$$(A \text{ AND } B) \text{ AND } C \longleftrightarrow A \text{ AND } (B \text{ AND } C) \qquad \text{(associativity of AND)} \tag{3.8}$$

$$\mathbf{T} \text{ AND } A \longleftrightarrow A \qquad \text{(identity for AND)}$$

$$\mathbf{F} \text{ AND } A \longleftrightarrow \mathbf{F} \qquad \text{(zero for AND)}$$

$$A \text{ AND } (B \text{ OR } C) \longleftrightarrow (A \text{ AND } B) \text{ OR } (A \text{ AND } C) \qquad \text{(distributivity of AND over OR)} \tag{3.9}$$

Associativity (3.8) 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.

Unlike arithmetic rules for numbers, there is also a distributivity law for "sums" over "products:"

$$A \text{ OR } (B \text{ AND } C) \longleftrightarrow (A \text{ OR } B) \text{ AND } (A \text{ OR } C) \qquad \text{(distributivity of OR over AND)} \tag{3.10}$$

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

$$A \text{ AND } A \longleftrightarrow A \qquad \text{(idempotence for AND)}$$

$$A \text{ AND } \overline{A} \longleftrightarrow \mathbf{F} \qquad \text{(contradiction for AND)} \tag{3.11}$$

$$\text{NOT}(\overline{A}) \longleftrightarrow A \qquad \text{(double negation)} \tag{3.12}$$

There are a corresponding set of equivalences for OR which we won't bother to list, except for validity rule (3.13) for OR:

$$A \text{ OR } \overline{A} \longleftrightarrow \mathbf{T} \qquad \text{(validity for OR)} \tag{3.13}$$

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

$$\text{NOT}(A \text{ AND } B) \longleftrightarrow \overline{A} \text{ OR } \overline{B} \qquad \text{(De Morgan for AND)} \tag{3.14}$$

$$\text{NOT}(A \text{ OR } B) \longleftrightarrow \overline{A} \text{ AND } \overline{B} \qquad \text{(De Morgan for OR)} \tag{3.15}$$

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

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

$$\text{NOT}((A \text{ AND } B) \text{ OR } (A \text{ AND } C)). \tag{3.16}$$

into a full DNF.

We start by applying De Morgan's Law for OR (3.15) to (3.16) 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 De Morgan's Law for AND (3.14) to the two innermost AND-terms, gives

$$(\overline{A} \text{ OR } \overline{B}) \text{ AND } (\overline{A} \text{ OR } \overline{C}). \tag{3.17}$$

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

Now we will repeatedly apply (3.9), distributivity of AND over OR, to turn (3.17) into a DNF. To start, we'll distribute $(\overline{A} \text{ OR } \overline{B})$ 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 of QOR now allows dropping the parentheses grouping the AND-clauses to yield the following DNF for (3.16):

$$\overline{A} \text{ OR } (\overline{B} \text{ AND } \overline{A}) \text{ OR } (\overline{A} \text{ AND } \overline{C}) \text{ OR } (\overline{B} \text{ AND } \overline{C}). \tag{3.18}$$

The penultimate step is to turn this DNF into a full DNF. This can be done separately for ech AND-clause. We'll illustrate how using the second AND-clause $(\overline{B} \text{ AND } \overline{A})$. This clause needs to mention C to be in full 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 in $(C \text{ OR } \overline{C})$ yields the full DNF

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

Doing the same thing to the other AND-clauses in (3.18) finally gives a full DNF for (3.5):

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

The final step is to use commutativity to sort the variables within the AND-clauses and then sort the AND-clauses themselves, followed by applying OR-idempotence as needed to remove duplicate AND-clauses. This finally yields a sorted full DNF without duplicates which is called a *canonical 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 the general strategy for applying the axioms above to any given propositional formala to derive an equivalent canonical DNF. This proves:

Theorem 3.4.2. *Using the equivalences listed above, any propositional formula can be proved equivalent to a canonical form.*

What has this got to do with equivalence? That's easy: to prove that two formulas are equivalent, convert them both to canonical forms over the set of variables that appear in at least one of the formulas—call these the *combined variables*. Now if two formulas are equivalent to the same canonical form then the formula are certainly equivalent. Conversely, the way we read off a full disjunctive normal form from a truth table actually yields a canonical form. So if two formulas are equivalent, they will have the same truth table over the combined variables, and therefore they will have the same canonical form. This proves

Theorem 3.4.3 (Completeness of the propositional equivalence axioms)**.** *Two propositional formula are equivalent iff they can be proved equivalent using the equivalence axioms listed above.*

Notice that the same approach could be taken used CNF instead of DNF canonical forms.

The benefit of the axioms is that they allow some ingenious proofs of equivalence that may involve much less effort than the truth table method. Moreover,

Theorem 3.4.3 reassures us that the axioms are guaranteed to provide a proof of every equivalence, which is a great punchline for this section.

But we don't want to mislead you: the guaranteed proof involves deriving canonical forms, and canonical forms are essentially copies of truth tables. There is no reason to expect algebraic proofs of equivalence to be any easier in general than conversion to canonical form, which means algebraic proofs will generally be no 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 because decrypting coded messages would 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 9.12.

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 are generally much less effective.

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.[2] 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

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

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

$$"5x^2 - 7 = 0"$$

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.

[2] **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.

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:

If you can solve any problem we come up with,
then you get an A for the course. (3.19)

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

you can solve *every* problem we come up with, (3.20)

or maybe

you can solve *at least one* problem we come up with. (3.21)

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 (3.19) can be written as follows:

$(\forall x \in \text{Probs. Solves}(x))$ IMPLIES G,	for (3.20),
$(\exists x \in \text{Probs. Solves}(x))$ IMPLIES G.	for (3.21).

3.6.2 Mixing Quantifiers

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

> Goldbach's Conjecture 1.1.6: 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 an instance of a general equivalence that holds between predicate formulas:

$$\text{NOT}(\forall x.\, P(x)) \quad \text{is equivalent to} \quad \exists x.\, \text{NOT}(P(x)). \tag{3.22}$$

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

$$\text{NOT}(\exists x.\, P(x)) \quad \text{is equivalent to} \quad \forall x.\, \text{NOT}(P(x)). \tag{3.23}$$

Note that the equivalence (3.23) follows directly by negating both sides the equivalence (3.22).

The general principle is that *moving a* NOT *to the other side of an "∃" changes it into "∀," and* vice versa.

These equivalences are called *De Morgan's Laws for Quantifiers* because they can be understood as applying De Morgan's Laws for propositional formulas to an infinite sequence of AND's and OR's. For example, we can explain (3.22) by

supposing the domain of discourse is $\{d_0, d_1, \ldots, d_n, \ldots\}$. Then $\exists x.\ \text{NOT}(P(x))$ means the same thing as the infinite OR:

$$\text{NOT}(P(d_0))\ \text{OR}\ \text{NOT}(P(d_1))\ \text{OR}\ \cdots\ \text{OR}\ \text{NOT}(P(d_n))\ \text{OR}\ \ldots. \tag{3.24}$$

Applying De Morgan's rule to this infinite OR yields the equivalent formula

$$\text{NOT}[P(d_0)\ \text{AND}\ P(d_1)\ \text{AND}\ \cdots\ \text{AND}\ P(d_n)\ \text{AND}\ \ldots]. \tag{3.25}$$

But (3.25) means the same thing as

$$\text{NOT}[\forall x.\ P(x)].$$

This explains why $\exists x.\ \text{NOT}(P(x))$ means the same thing as $\text{NOT}[\forall x.\ P(x)]$, which confirms(3.22).

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 (3.22) that gives the rule for negating a universal quantifier means that the following formula is valid:

$$\text{NOT}(\forall x.\ P(x))\ \text{IFF}\ \exists x.\ \text{NOT}(P(x)). \tag{3.26}$$

Another useful example of a valid assertion is

$$\exists x \forall y.\ P(x, y)\ \text{IMPLIES}\ \forall y \exists x.\ P(x, y). \tag{3.27}$$

Here's an explanation why this is valid:

> Let D be the domain for the variables and P_0 be some binary predicate[3] on D. We need to show that if
>
> $$\exists x \in D.\, \forall y \in D.\, P_0(x, y) \tag{3.28}$$
>
> holds under this interpretation, then so does
>
> $$\forall y \in D\, \exists x \in D.\, P_0(x, y). \tag{3.29}$$
>
> So suppose (3.28) is true. Then by definition of $\exists$, this means that some element $d_0 \in D$ has the property that
>
> $$\forall y \in D.\, P_0(d_0, y).$$

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

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 (3.29) means, so we've proved that (3.29) holds under this interpretation, as required.

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 (3.27), 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 (3.27) into English and then appeal to the meaning, in English, of "for all" and "there exists" as justification.

In contrast to (3.27), the formula

$$\forall y \exists x.\, P(x, y) \text{ IMPLIES } \exists x \forall y.\, P(x, y). \tag{3.30}$$

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

[21]

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.

Problem 3.5.
Sloppy Sam is trying to prove a certain proposition P. He defines two related propositions Q and R, and then proceeds to prove three implications:

$$P \text{ IMPLIES } Q, \qquad Q \text{ IMPLIES } R, \qquad R \text{ IMPLIES } P.$$

He then reasons as follows:

If Q is true, then since I proved (Q IMPLIES R), I can conclude that R is true. Now, since I proved (R IMPLIES P), I can conclude that P is true. Similarly, if R is true, then P is true and so Q is true. Likewise, if P is true, then so are Q and R. So any way you look at it, all three of P, Q and R are true.

(a) Exhibit truth tables for

$$(P \text{ IMPLIES } Q) \text{ AND } (Q \text{ IMPLIES } R) \text{ AND } (R \text{ IMPLIES } P) \qquad (*)$$

and for

$$P \text{ AND } Q \text{ AND } R. \qquad (**)$$

Use these tables to find a truth assignment for P, Q, R so that (*) is **T** and (**) is **F**.

(b) You show these truth tables to Sloppy Sam and he says "OK, I'm wrong that P, Q and R all have to be true, but I still don't see the mistake in my reasoning. Can you help me understand my mistake?" How would you explain to Sammy where the flaw lies in his reasoning?

Problems for Section 3.2

Class Problems

Problem 3.6.

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 3.7.

As in Problem 3.6, a digital circuit is called an $(n+1)$-bit *half-adder* when it has with $n+2$ inputs

$$a_n, \ldots, a_1, a_0, b$$

and $n+2$ outputs

$$c, s_n, \ldots, s_1, s_0.$$

The input-output specification of the half-adder is that, if the 0-1 values of inputs $a_n, \ldots, a_1, a_0$ are taken to be the $(n+1)$-bit binary representation of an integer k

then the 0-1 values of the outputs $c, s_n, \ldots, s_1, s_0$ are supposed to be the $(n+2)$-bit binary representation of $k+b$.

For example suppose $n = 2$ and the values of $a_2a_1a_0$ were `101`. This is the binary representation of $k = 5$. Now if the value of b was `1`, then the output should be the 4-bit representation of $5 + 1 = 6$. Namely, the values of $cs_2s_1s_0$ would be `0110`.

There are many different circuit designs for half adders. The most straighforward one is the "ripple carry" design described in Problem 3.6. We will now develop a different design for a half-adder circuit called a *parallel-design* or "look-ahead carry" half-adder. This design works by computing the values of higher-order digits for both a carry of 0 and a carry of 1, *in parallel*. Then, when the carry from the low-order digits finally arrives, the pre-computed answer can be quickly selected.

We'll illustrate this idea by working out a parallel design for an $(n+1)$-bit half-adder.

Parallel-design half-adders are built out of parallel-design circuits called *add1*-modules. The input-output behavior of an add1-module is just a special case of a half-adder, where instead of an adding an input b to the input, the add1-module *always* adds 1. That is, an $(n+1)$-bit add1-module has $(n+1)$ binary inputs

$$a_n, \ldots, a_1, a_0,$$

and $n+2$ binary outputs

$$c\ p_n, \ldots, p_1, p_0.$$

If $a_n \ldots a_1a_0$ are taken to be the $(n+1)$-bit representation of an integer k then $cp_n \ldots p_1p_0$ is supposed to be the $(n+2)$-bit binary representation of $k+1$.

So a 1-bit add1-module just has input a_0 and outputs c, p_0 where

$$p_0 ::= a_0 \ \text{XOR}\ 1, \text{(or more simply, } p_0 ::= \text{NOT}(a_0)),$$
$$c ::= a_0.$$

In the ripple-carry design, a double-size half-adder with $2(n+1)$ inputs takes twice as long to produce its output values as an $(n+1)$-input ripple-carry circuit. With parallel-design add1-modules, a double-size add1-module produces its output values nearly as fast as a single-size add1-modules. To see how this works, suppose the inputs of the double-size module are

$$a_{2n+1}, \ldots, a_1, a_0$$

and the outputs are

$$c, p_{2n+1}, \ldots, p_1, p_0.$$

We will build the double-size add1-module by having two single-size add1-modules work in parallel. 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.

(a) Write a formula for the carry c of the double-size add1-module *solely* in terms of carries $c_{(1)}$ and $c_{(2)}$ of the single-size add1-modules.

(b) Complete the specification of the double-size add1-module by writing propositional formulas for the remaining outputs p_{n+i} for $1 \leq i \leq n+1$. The formula for p_{n+i} should only involve the variables a_{n+i}, r_{i-1} and $c_{(1)}$.

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

(d) The speed or *latency* of a circuit is determined by the largest number of gates on any path from an input to an output. In an n-bit ripple carry circuit(Problem 3.6), there is a path from an input to the final carry output that goes through about $2n$ gates. In contrast, parallel half-adders are exponentially faster than ripple-carry half-adders. Confirm this by determining the largest number of propositional operations, that is, gates, on any path from an input to an output of an n-bit add1-module. (You may assume n is a power of 2.)

Exam Problems

Problem 3.8.

Claim. *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?

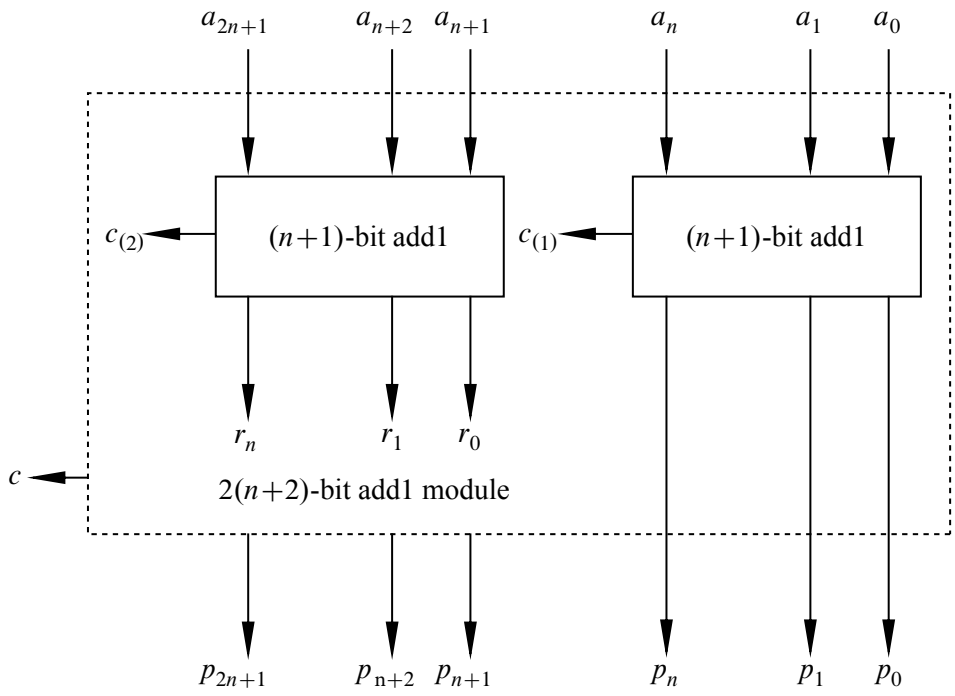

Figure 3.1 Structure of a Double-size *add1* Module.

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

Problem 3.9.
An n-bit AND-circuit has 0-1 valued inputs $a_0, a_1, \ldots, a_{n-1}$ and one output c whose value will be

$$c = a_0 \text{ AND } a_1 \text{ AND } \cdots \text{ AND } a_{n-1}.$$

There are various ways to design an n-bit AND-circuit. A *serial* design is simply a series of AND-gates, each with one input being a circuit input a_i and the other input being the output of the previous gate as shown in Figure 3.2.

We can also use a *tree* design. A 1-bit tree design is just a wire, that is $c ::= a_1$. Assuming for simplicity that n is a power of two, an n-input tree circuit for $n > 1$ simply consists of two $n/2$-input tree circuits whose outputs are AND'd to produce output c, as in Figure 3.3. For example, a 4-bit tree design circut is shown in Figure 3.4.

(a) How many AND-gates are in the n-input serial circuit?

(b) The "speed" or *latency* of a circuit is the largest number of gates on any path from an input to an output. Briefly explain why the tree circuit is *exponentially faster* than the serial circuit.

(c) Assume n is a power of two. Prove that the n-input tree circuit has $n - 1$ AND-gates.

Problems for Section 3.3

Practice Problems

Problem 3.10.
Indicate the correct choices:

(a) A formula is satisfiable iff its negation is

- not satisfiable
- also satisfiable
- valid
- not valid

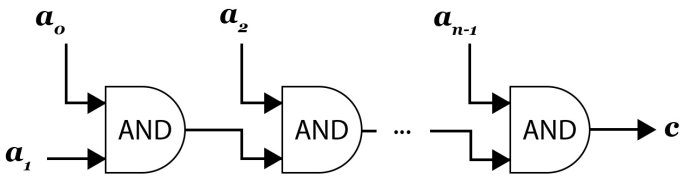

Figure 3.2 A serial AND-circuit.

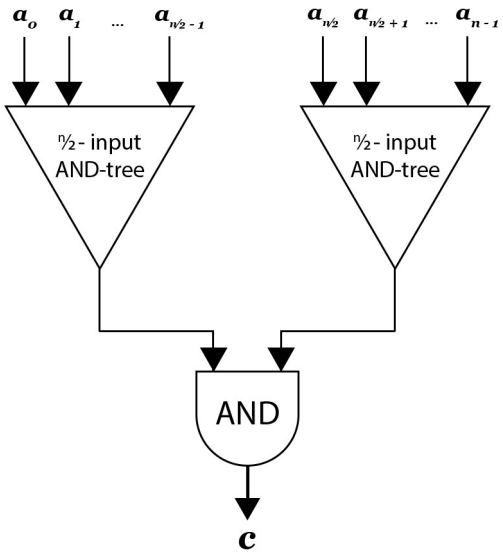

Figure 3.3 An n-bit AND-tree circuit.

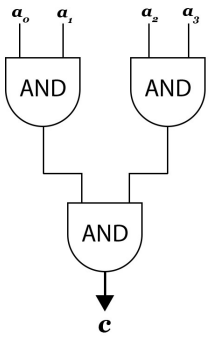

Figure 3.4 A 4-bit AND-tree circuit.

(b) A formula is valid iff its negation is

- not valid
- also valid
- satisfiable
- not satisfiable

(c) Formula F is equivalent to formula G iff

- F IFF G is valid
- F IFF NOT(G) is not valid
- F XOR G satisfiable
- F XOR G is not satisfiable

Problem 3.11.
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 3.12.
Show truth tables that verify the equivalence of the following two propositional formulas

$$
\begin{aligned}
&(P \text{ XOR } Q),\\
&\text{NOT}(P \text{ IFF } Q).
\end{aligned}
$$

Problem 3.13.
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 3.14.
Prove by truth table that OR distributes over AND, namely,

$$P \text{ OR } (Q \text{ AND } R) \quad \text{is equivalent to} \quad (P \text{ OR } Q) \text{ AND } (P \text{ OR } R) \tag{3.31}$$

Exam Problems

Problem 3.15.
The formula

$$\begin{array}{c} \text{NOT}(\overline{A} \text{ IMPLIES } B) \text{ AND } A \text{ AND } C \\ \text{IMPLIES} \\ D \text{ AND } E \text{ AND } F \text{ AND } G \text{ AND } H \text{ AND } I \text{ AND } J \text{ AND } K \text{ AND } L \text{ AND } M \end{array}$$

turns out to be valid.

(a) Explain why verifying the validity of this formula *by truth table* would be very hard for one person to do with pencil and paper (no computers).

(b) Verify that the formula is valid, reasoning by cases according to the truth value of A.

Proof. **Case**: (A is **True**).

Case: (A is **False**).

■

Class Problems

Problem 3.16. **(a)** Verify by truth table that

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

is valid.

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

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

P 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 such that the set $P_1, \ldots, P_k$ is *not* consistent iff S is valid.

Problem 3.17.
This problem[4] examines whether the following specifications are *satisfiable*:

1. If the file system is not locked, then...
   (a) new messages will be queued.
   (b) new messages will be sent to the messages buffer.
   (c) the system is functioning normally, and conversely, if the system is functioning normally, then the file system is not locked.
2. If new messages are not queued, then they will be sent to the messages buffer.
3. New messages will not be sent to the message buffer.

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

L ::= file system locked,
Q ::= new messages are queued,
B ::= new messages are sent to the message buffer,
N ::= system functioning normally.

[4]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 3.18.
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).

Problem 3.19.
The five-variable propositional formula

$$P ::= (A \text{ AND } B \text{ AND } \overline{C} \text{ AND } D \text{ AND } \overline{E}) \text{ OR } (\overline{A} \text{ AND } B \text{ AND } \overline{C} \text{ AND } \overline{E})$$

is in Disjunctive Normal Form with two "AND-of-literal" clauses.

(a) Find a **Full** Disjunctive Normal Form that is equivalent to P, and explain your reasoning.

Hint: Can you narrow in on the important parts of the truth table without writing all of it? Alternatively, can you avoid the truth table altogether?

(b) Let C be a **Full Con**junctive Normal Form that is equivalent to P. Assume that C has been simplified so that none of its "OR-of-literals" clauses are equivalent

to each other. How many clauses are there in C? (Please **don't** try to write out any of these clauses.)

Briefly explain your answer.

Class Problems

Problem 3.20.
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 3.21.
Explain a simple way to obtain a conjunctive normal form (CNF) for a propositional formula directly from a disjunctive normal form (DNF) for its complement.

Hint: DeMorgan's Law does most of the job. Try working an illustrative example of your choice before describing the general case.

Problem 3.22.
Let P be the proposition depending on propositional variable A, B, C, D whose truth values for each truth assignment to A, B, C, D are given in the table below. Write out both a disjunctive and a conjunctive normal form for P.

A	B	C	D	P
T	**T**	**T**	**T**	**T**
T	**T**	**T**	**F**	**F**
T	**T**	**F**	**T**	**T**
T	**T**	**F**	**F**	**F**
T	**F**	**T**	**T**	**T**
T	**F**	**T**	**F**	**T**
T	**F**	**F**	**T**	**T**
T	**F**	**F**	**F**	**T**
F	**T**	**T**	**T**	**T**
F	**T**	**T**	**F**	**F**
F	**T**	**F**	**T**	**T**
F	**T**	**F**	**F**	**F**
F	**F**	**T**	**T**	**F**
F	**F**	**T**	**F**	**F**
F	**F**	**F**	**T**	**T**
F	**F**	**F**	**F**	**T**

Homework Problems

Problem 3.23.
Use the equivalence axioms of Section 3.4.2 to convert the formula

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

(a) ...to disjunctive (OR of AND's) form,

(b) ...to conjunctive (AND of OR's) form.

Exam Problems

Problems for Section 3.5

Homework Problems

Problem 3.24.
The circuit-SAT problem is the problem of determining, for any given digital circuit with one output wire, whether there are truth values that can be fed into the circuit input wires which will lead the circuit to give output **T**.

It's easy to see that any efficient way of solving the circuit-SAT problem would yield an efficient way to solve the usual SAT problem for propositional formulas (Section 3.5). Namely, for any formula F, just construct a circuit C_F using that computes the values of the formula. Then there are inputs for which C_F gives output true iff F is satisfiable. Constructing C_F from F is easy, using a binary gate in C_F for each propositional connective in F. So an efficient circuit-SAT procedure leads to an efficient SAT procedure.

Conversely, there is a simple recursive procedure that will construct, given C, a formula E_C that is equivalent to C in the sense that the truth value E_C and the output of C are the same for every truth assignment of the variables. The difficulty is that, in general, the "equivalent" formula E_C, will be *exponentially larger* than C. For the purposes of showing that satifiability of circuits and satisfiability of formulas take roughly the same effort to solve, spending an exponential time translating one problem to the other swamps any benefit in switching from one problem to the other.

So instead of a formula E_C that is equivalent to C, we aim instead for a formula F_C that is "equisatisfiable" with C. That is, there will be input values that make C output **True iff** there is a truth assignment that satisfies F_C. (In fact, F_C and C need not even use the same variables.) But now we make sure that the amount of computation needed to construct F_C is not much larger than the size of the circuit C. In particular, the size of F_C will also not be much larger than C.

The idea behind the construction of F_C is that, given any digital circuit C with binary gates and one output, we can assign a distinct variable to each wire of C. Then for each gate of C, we can set up a propositional formula that represents the constraints that the gate places on the values of its input and output wires. For example, for an AND gate with input wire variables P and Q and output wire variable R, the constraint proposition would be

$$(P \text{ AND } Q) \text{ IFF } R. \tag{3.32}$$

(a) Given a circuit C, explain how to easily find a formula F_C of size proportional to the number of wires in C such that F_C is satisfiable iff C gives output **T** for some set of input values.

(b) Conclude that any efficient way of solving SAT would yield an efficient way to solve circuit-SAT.

Problem 3.25.

A 3-conjunctive normal form (3CNF) formula is a conjunctive normal form (CNF)

formula in which each OR-term is an OR of at most 3 *literals* (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

- a 3CNF,
- has at most 24 times as many occurrences of variables as F, and
- is satisfiable iff F is satisfiable.

Note that we do *not* expect $\mathcal{C}(F)$ to be *equivalent* to F. We do know how to convert any F into an equivalent CNF formula, and this equivalent CNF formula will certainly be satisfiable iff F is. But in many cases, the smallest CNF formula equivalent to F may be *exponentially larger* than F instead of only 24 times larger. Even worse, there may not be any **3**CNF equivalent to F.

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

$$((P \text{ XOR } Q) \text{ XOR } R) \text{ OR } (\overline{P} \text{ AND } S) \tag{3.33}$$

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 (3.33) is satisfiable.

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

(c) Using the ideas illustrated in the previous parts, briefly explain how to construct $\mathcal{C}(F)$ for an arbitrary propositional formula F. (No need to fill in all the details for this part—a high-level description is fine.)

Problems for Section 3.6

Practice Problems

Problem 3.26.
For each of the following propositions:

(i) $\forall x\, \exists y.\ 2x - y = 0$,

(ii) $\forall x\, \exists y.\ x - 2y = 0$,

(iii) $\forall x.\, x < 10$ IMPLIES $(\forall y.\ y < x$ IMPLIES $y < 9)$,

(iv) $\forall x\, \exists y.\ [y > x \wedge \exists z.\ y + z = 100]$,

indicate which propositions are **False** when the variables range over:

(a) the nonnegative integers.

(b) the integers.

(c) the real numbers.

Problem 3.27.
Let $Q(x, y)$ be the statement

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

The domain 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.

Indicate which of the following expressions are 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 3.28.
Express each of the following statements using quantifiers, logical connectives, and/or the following predicates

$P(x)$**:** x is a monkey,

$Q(x)$**:** x is a 6.042 TA,

$R(x)$**:** x comes from the 23rd century,

$S(x)$**:** x likes to eat pizza,

where x ranges over all living things.

(a) No monkeys like to eat pizza.

(b) Nobody from the 23rd century dislikes eating pizza.

(c) All 6.042 TAs are monkeys.

(d) No 6.042 TA comes from the 23rd century.

(e) Does part (d) follow logically from parts (a), (b), (c)? If so, give a proof. If not, give a counterexample.

(f) Translate into English: $(\forall x)(R(x)$ OR $S(x)$ IMPLIES $Q(x))$.

(g) Translate into English:

$$[\exists x.\, R(x) \text{ AND NOT}(Q(x))] \text{ IMPLIES } \forall x.\, (P(x) \text{ IMPLIES } S(x)).$$

Problem 3.29.
Show that

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

is not valid by describing a counter-model.

Problem 3.30.
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 3.31.
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 3.32.
Which of the following are *valid*? For those that are not valid, desribe a counter-model.

(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$(S(x))$

Problem 3.33. (a) Verify that the propositional formula

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

is valid.

(b) The valid formula of part (a) leads to sound proof method: to prove that an implication is true, just prove that its converse is false.[5] For example, from elementary calculus we know that the assertion

> If a function is continuous, then it is differentiable

is false. This allows us to reach at the correct conclusion that its converse,

> If a function is differentiable, then it is continuous

is true, as indeed it is.
But wait a minute! The implication

> If a function is differentiable, then it is not continuous

is completely false. So we could conclude that its converse

[5] This problem was stimulated by the discussion of the fallacy in [4].

If a function is not continuous, then it is differentiable,

should be true, but in fact the converse is also completely false.

So something has gone wrong here. Explain what.

Class Problems

Problem 3.34.

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
>
> $$\{\text{Albert, Ben, Claire, David, Emily}\}.$$
>
> 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)

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

(b)

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

(c)

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

(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 3.35.
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 3.36.
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	$\exists z\ (xz = y)$	PREFIX(x, y)
x is a substring of y	$\exists u \exists v\ (uxv = y)$	SUBSTRING(x, y)
x is empty or a string of 0's	NOT(SUBSTRING$(1, x)$)	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 3.37.
In this problem we'll examine predicate logic formulas where the domain of discourse is $\mathbb{N}$. In addition to the logical symbols, the formulas may contain ternary predicate symbols A and M, where

$$A(k, m, n) \text{ means } k = m + n,$$
$$M(k, m, n) \text{ means } k = m \cdot n.$$

For example, a formula "Zero(n)" meaning that n is zero could be defined as

$$\text{Zero}(n) ::= A(n, n, n).$$

Having defined "Zero," it is now OK to use it in subsequent formulas. So a formula "Greater(m, n)" meaning $m > n$ could be defined as

$$\text{Greater}(m, n) ::= \exists k.\, \text{NOT}(\text{Zero}(k)) \text{ AND } A(m, n, k).$$

This makes it OK to use "Greater" in subsequent formulas.

Write predicate logic formulas using only the allowed predicates A, M that define the following predicates:

(a) Equal(m, n) meaning that $m = n$.

(b) One(n) meaning that $n = 1$.

(c) $n = i(m \cdot j + k^2)$

(d) Prime(p) meaning p is a prime number.

(e) Two(n) meaning that $n = 2$.

The results of part (e) will entend to formulas $\text{Three}(n), \text{Four}(n), \text{Five}(n), \ldots$ which are allowed from now on.

(f) $\text{Even}(n)$ meaning n is even.

(g) (Goldbach Conjecture) Every even integer $n \geq 4$ can be expressed as the sum of two primes.

(h) (Fermat's Last Theorem) Now suppose we also have

$$X(k, m, n) \quad \text{means } k = m^n.$$

Express the assertion that there are no positive integer solutions to the equation:

$$x^n + y^n = z^n$$

when $n > 2$.

(i) (Twin Prime Conjecture) There are infinitely many primes that differ by two.

Homework Problems

Problem 3.38.
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 3.39.
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 3.40. (a) Translate the following sentence into a predicate formula:

> There is a student who has e-mailed at most n 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,
- $E(x, y)$, meaning that "x has sent e-mail to y."

(b) Explain how you would use your predicate formula (or some variant of it) to express the following two sentences.

1. There is a student who has emailed at least n other people in the class, besides possibly himself.
2. There is a student who has emailed exactly n other people in the class, besides possibly himself.

Problem 3.41.
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

$$(\forall x.P(x)) \text{ AND } (\forall x.Q(x)) \tag{3.34}$$

can be rewritten as the equivalent formula

$$(\forall y.P(y)) \text{ AND } (\forall x.Q(x)), \tag{3.35}$$

which more clearly shows that the separate occurrences of $\forall x$ in (3.34) 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 (3.34) violates the unique variable convention because "$\forall x$" occurs twice, but formula (3.35) is OK.

A further example is the formula

$$[\forall x\, \exists y.\, P(x) \text{ AND } Q(x,y)] \text{ IMPLIES } (\exists x.\, R(x,z)) \text{ OR } \exists x\, \forall z.\, S(x,y,w,z). \tag{3.36}$$

Formula (3.36) 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 when we renamed the first two occurrences of x in (3.34) into y's to obtain the equivalent formula (3.35).

Rename occurrences of variables in (3.36) to obtain an equivalent formula that satisfies the unique variable convention. Try to rename as few occurrences as possible.

A general procedure for converting a predicate formula into unique variable format is described in Problem 7.32.

Problem 3.42.

A predicate formula is in *prenex form* when all its quantifiers come at the beginning. For example,

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

is in prenex form.

It turns out that every formula is equivalent to one in prenex form. Here's a simple recipe for converting a formula into an equivalent prenex form:

1. Rename variables to satisfy the *variable convention* described in Problem 3.41,
2. Convert all connectives between formulas with a quantifier into AND, OR, NOT,
3. Use De Morgan Laws for negated quantifiers (Section 3.6.5) to push negations "below" all quantifiers,
4. Pull out all quantifiers to the front of the formula in any order that preserves nesting. That is, if $\exists x$ occurs in a subformula starting with $\forall y$, then $\forall y$ has to come before $\exists x$ when the quantifiers are moved to the front.

Use this method to find a prenex form for

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

Problem 3.43.

Let $\tilde{\mathbf{0}}$ be a constant symbol, **next** and **prev** be function symbols taking one argument.

We can picture any model for these symbols by representing domain elements as points. The model must interpret $\tilde{\mathbf{0}}$ as some point e_0. It also interprets the symbols **next** and **prev** as total functions *nextf* and *prevf* on the domain. We can picture the functions by having an arrow labelled **next** go out from each point e into the point *nextf*(e), and an arrow labelled **prev** go out from each e into *prevf*(e). In particular,

*Every point has exactly one **next**-arrow, and exactly one **prev**-arrow, going out of it.*

The aim of this problem is to develop a series of predicate formulas using just the symbols $\tilde{\mathbf{0}}$, **next** and **prev** such that every model satisfying these formulas will contain contain a copy of the nonnegative integers $\mathbb{N}$, with *nextf* acting on the copy as the "$+1$" or *successor function* and *prevf* acting as the "-1" or *predecessor function*.

More precisely, the "copy" of $\mathbb{N}$ in a model will look like an infinite sequence of distinct points starting with e_0, with a **next**-arrow going from each point to the next

in the sequence:

$$e_0 \overset{\textbf{next}}{\longrightarrow} e_1 \overset{\textbf{next}}{\longrightarrow} e_2 \overset{\textbf{next}}{\longrightarrow} \ldots \overset{\textbf{next}}{\longrightarrow} e_n \overset{\textbf{next}}{\longrightarrow} \ldots$$

so that *nextf* acts like plus one. We also want *prevf* to act like minus one: whenever a **next**-arrow goes into an element e, then the **prev**-arrow out of e goes back to the beginning of the **next**-arrow:

$$e_0 \underset{\textbf{prev}}{\overset{\textbf{next}}{\rightleftarrows}} e_1 \underset{\textbf{prev}}{\overset{\textbf{next}}{\rightleftarrows}} e_2 \underset{\textbf{prev}}{\overset{\textbf{next}}{\rightleftarrows}} \ldots \underset{\textbf{prev}}{\overset{\textbf{next}}{\rightleftarrows}} e_n \underset{\textbf{prev}}{\overset{\textbf{next}}{\rightleftarrows}} \ldots . \tag{3.37}$$

Not shown is a further **prev**-arrow self-loop from e_0 to e_0, reflecting the convention for nonnegative integers that subtracting from zero has no effect.

There is a simple way to express this requirement as a predicate formula:

$$\forall x.\, \textbf{prev}(\textbf{next}(x)) = x\ . \tag{3.38}$$

Formula (3.38) means that the **prev**-arrow out of any point e goes back to the beginning of any **next**-arrow into e. Of course this will not be possible if there is more than one **next**-arrow into e.

There are some standard terms called *numerals* used to describe the elements $e_0, e_1, \ldots$ in (3.37). Namely, the numeral for e_0 will be $\tilde{\mathbf{0}}$, and the numeral for e_n will be the application n times of **next** to $\tilde{\mathbf{0}}$. For example, the numeral for three would be:

$$\textbf{next}(\textbf{next}(\textbf{next}(\tilde{\mathbf{0}}))). \tag{3.39}$$

We'll refer to the numeral for e_n as $\widetilde{\mathrm{n}}$. So $\widetilde{3}$ refers to the term (3.39).

But we don't quite have the desired situation pictured in (3.37): there is nothing so far that prevents all the numerals having the same value. In other words, the formulas above are consistent with the formula $\textbf{next}(\tilde{\mathbf{0}}) = \tilde{\mathbf{0}}$. (You should check this yourself right now.) We might try to fix this by requiring that no **next**-arrow can begin and end at the same element, but there still could be a pair of numeral values, each of which was *nextf* of the other. That is, the model might satisfy

$$\widetilde{2} ::= \textbf{next}(\textbf{next}(\tilde{\mathbf{0}})) = \tilde{\mathbf{0}}.$$

We could go on to forbid such length two cycles of **next**-arrows, but then there might be a cycle of three:

$$\widetilde{3} ::= \textbf{next}(\textbf{next}(\textbf{next}(\tilde{\mathbf{0}}))) = \tilde{\mathbf{0}},$$

and so on. Fortunately, something we want to do anyway will fix this potential problem with the numeral values: we haven't yet provided a formula that will make the interpretion e_0 of $\tilde{\mathbf{0}}$ behave like zero.

(a) Write a predicate formula expressing the fact that the value of $\tilde{\mathbf{0}}$ is not plus-one of anything. Also, by convention, subtracting one from zero has no effect.

The formulas of part (a) and (3.38) together imply that the numeral values will yield the copy (3.37) of $\mathbb{N}$ we want. To verify this, we just need to show that the values of all the numerals are distinct.

(b) Explain why two different numerals must have different values in any model satisfying (3.38) and part (a).

Hint: It helps to describe the meaning of (3.38) by what it says about arrows. There is also an explanation based on the Well Ordering Principle.

So we have verified that any model satisfying (3.38) and the formula of part (a) has the desired copy of $\mathbb{N}$.

The distinction between formulas that are *true* over the domain $\mathbb{N}$ and formulas that are valid, that is true over *all domains* can be confusing on first sight. To highlight the distinction, it is worth seeing that the formula of part (a) together with (3.38) do *not* ensure that their models consist *solely* of a copy of $\mathbb{N}$.

For example, a model might consist of two separate copies of $\mathbb{N}$. We can stifle this particular possibility of extra copies of $\mathbb{N}$ pretty easily. Any such copy has to start with a zero-like element, namely one that is not in the range of *nextf*. So we just assert that there is the only zero-like element.

(c) Write a formula such that any model has only one copy of $\mathbb{N}$.

But the additional axiom of part (c) still leaves room for models that, besides having a copy of $\mathbb{N}$, also have "extra stuff" with weird properties.

(d) Describe a model that satisfies the formula of part (c) along with (3.38), part (a) and also satisfies

$$\exists x.\, x = \mathbf{next}(x). \tag{3.40}$$

Supplemental Part

(e) Prove that

$$\forall x \neq \tilde{\mathbf{0}}.\, \mathbf{next}(\mathbf{prev}(x)) = x. \tag{3.41}$$

Exam Problems

Problem 3.44.
Consider these five domains of discourse, in order:

$\mathbb{N}$ (nonnegative integers), $\mathbb{Z}$ (integers), $\mathbb{Q}$ (rationals), $\mathbb{R}$ (reals), $\mathbb{C}$ (complex numbers).

For each of the logic formulas below, indicate the first of these domains where the formula is true, or state "**none**" if it is not true in any of them. Please briefly explain each one.

i. $\forall x \exists y.\ y = 3x$

ii. $\forall x \exists y.\ 3y = x$

iii. $\forall x \exists y.\ y^2 = x$

iv. $\forall x \exists y.\ y < x$

v. $\forall x \exists y.\ y^3 = x$

vi. $\forall x \neq 0.\, \exists y, z.\ y \neq z$ AND $y^2 = x = z^2$

Problem 3.45.

The following predicate logic formula is invalid:

$$[\forall x, \exists y. P(x, y) \text{ IMPLIES } \exists y, \forall x. P(x, y) \tag{3.42}$$

Indicate which of the following are counter models for (3.42), and briefly explain.

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 3.46.

Some (but not necessarily all) students from a large class will be lined up left to right. There will be at least two students in the line. Translate each of the following assertions into predicate formulas, using quantifiers $\exists$ and $\forall$ and any logical operators like AND, OR, NOT, etc., with the set of students in the class as the domain of discourse. The only predicates you may use about students 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. (A student is not considered to be "to the left of" themself, so $F(x, x)$ is always false.)

As a worked example, the predicate "there are at least two students after x" may be written as

$$\exists y\, \exists z.\ \text{NOT}(y = z)\ \text{AND}\ (F(x, y)\ \text{AND}\ F(x, z)).$$

In your answers you may use earlier predicates in later parts, even if you did not solve the earlier parts. So your answer to part (c) might refer to inline(x) and/or first(x), for example.

(a) Student x is in the line. Call this predicate inline(x).

Hint: Since there are at least 2 students in the line, inline(x) means x is to the left or right of someone else.

(b) Student x is first in line. Call this predicate first(x).

(c) Student x is immediately to the right of student y. Call this predicate isnext(x, y).

Hint: No other student can be between x and y.

(d) Student x is second in line.

Problem 3.47.

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) $[x = y + 1]$.

Hint: If an integer is bigger than y, then it must be $\geq x$.

(c) $x = 3$.

Problem 3.48.

Predicate Formulas whose only predicate symbol is equality are called "pure equality" formulas. For example,

$$\forall x\, \forall y.\ x = y \qquad \text{(1-element)}$$

is a pure equality formula. Its meaning is that there is exactly one element in the domain of discourse.[6] Another such formula is

$$\exists a\, \exists b\, \forall x.\ x = a \text{ OR } x = b. \qquad (\leq 2\text{-elements})$$

Its meaning is that there are at most two elements in the domain of discourse.

A formula that is not a pure equality formula is

$$x \leq y. \qquad (\textbf{not}\text{-pure})$$

Formula (**not**-pure) uses the less-than-or-equal predicate $\leq$ which is *not* allowed.[7]

(a) Describe a pure equality formula that means that there are *exactly* two elements in the domain of discourse.

(b) Describe a pure equality formula that means that there are *exactly* three elements in the domain of discourse.

Problem 3.49. (a) A predicate R on the nonnegative integers is true *infinitely often (i.o.)* when $R(n)$ is true for infinitely many $n \in \mathbb{N}$.

We can express the fact that R is true i.o. with a formula of the form:

$$\mathbf{Q}_1\ \ \mathbf{Q}_2.\ R(n),$$

where $\mathbf{Q}_1, \mathbf{Q}_2$ are quantifiers from among

$$\begin{array}{llll} \forall n, & \exists n, & \forall n \geq n_0, & \exists n \geq n_0, \\ \forall n_0, & \exists n_0, & \forall n_0 \geq n, & \exists n_0 \geq n, \end{array}$$

and n, n_0 range over nonnegative integers.

Identify the proper quantifers: $\mathbf{Q}_1$____, $\mathbf{Q}_2$____

(b) A predicate S on the nonnegative integers is true *almost everywhere (a.e.)* when $S(n)$ is false for only finitely many $n \in \mathbb{N}$.

We can express the fact that S is true a.e. with a formula of the form

$$\mathbf{Q}_3\ \ \mathbf{Q}_4.\ S(n),$$

where $\mathbf{Q}_3, \mathbf{Q}_4$ are quantifiers from those above.

Identify the proper quantifers: $\mathbf{Q}_3$____, $\mathbf{Q}_4$____

[6] Remember, a domain of discourse is not allowed to be empty.

[7] In fact, formula (**not**-pure) only makes sense when the domain elements are ordered, while pure equality formulas make sense over every domain.

Problem 3.50.
Let $f : \mathbb{N} \to \mathbb{R}$ be a real-valued total function. A *limit point* of f is a real number $r \in \mathbb{R}$ such that $f(n)$ is close to r for *infinitely many* n, where "close to" means within distance ϵ for whatever positive real number ϵ you may choose.

We can express the fact that r is a limit point of f with a logical formula of the form:

$$\mathbf{Q_0}\ \mathbf{Q_1}\ \mathbf{Q_2}.\ |f(n) - r| \leq \epsilon,$$

where $\mathbf{Q_0}, \mathbf{Q_1}, \mathbf{Q_2}$ is a sequence of three quantifiers from among:

$$\begin{array}{cccc} \forall n, & \exists n, & \forall n \geq n_0, & \exists n \geq n_0. \\ \forall n_0, & \exists n_0, & \forall n_0 \geq n, & \exists n_0 \geq n. \\ \forall \epsilon \geq 0, & \exists \epsilon \geq 0, & \forall \epsilon > 0, & \exists \epsilon > 0. \end{array}$$

Here the n, n_0 range over nonnegative integers, and ϵ ranges over real numbers.

Identify the proper quantifers:

$$\mathbf{Q_0}$$

$$\mathbf{Q_1}$$

$$\mathbf{Q_2}$$

Problem 3.51.
Let $f : \mathbb{R} \to \mathbb{R}$ be a total function on the real numbers. The limit $\lim_{x \to r} f(x)$ of $f(x)$ as x approaches some real number r is a number $a \in \mathbb{R}$ such that $f(x)$ will be as close as you want (within distance $\epsilon > 0$) to a as long as x is close enough (within distance $\delta > 0$) to r.

This can be expressed by a logical formula of the form

$$Q_1\, \epsilon \in \mathbb{R}^+\ Q_2\, x \in \mathbb{R}\ Q_3\, \delta \in \mathbb{R}^+.\ |x - r| \leq \alpha \text{ IMPLIES } |f(x) - a| \leq \beta,$$

where Q_1, Q_2, Q_3 are quantifiers $\forall, \exists$ and α, β are each one δ or ϵ. Indicate which:

$$\begin{array}{rcll} Q_1 & \text{is} & \forall & \exists \\ Q_2 & \text{is} & \forall & \exists \\ Q_3 & \text{is} & \forall & \exists \\ \alpha & = & \delta & \epsilon \\ \beta & = & \delta & \epsilon \end{array}$$