Chapter 1

Logic

A major topic in most discrete structures books, this one included, is proof techniques. Proofs are arguments, arguments are built from inferences, inferences are implications, and an implication is an operator that applies to logical propositions. To understand proofs, we need to start with logic.

1.1 What is Logic?

logic

In brief, *logic* is the art of intellectual persuasion, and comes in many forms. For our purposes, we consider the branches known as *philosophical* and *mathematical* logic.

philosophical logic

Definition 1: Philosophical Logic

Philosophical Logic is the study of thought and reasoning as expressed in natural languages.

mathematical logic

Definition 2: Mathematical Logic

Mathematical Logic is the use of formal languages to represent reasoning and computation.

Neither of these definitions is universally accepted by philosophers or mathematicians, but they are adequate to show that logic as used in computer science is not easily pigeonholed.[1] In this book, we will use concepts from both categories.

propositional logic

Our path to proofs will begin with *propositional logic*, also known as *sentential calculus*.[2]. As the name suggests, the key idea is that of propositions, which are logical formulae, often compounded with logical operators, that evaluate to the values of true or false. Of particular use to us will be *first-order logic*. In FOL (also known as *(first-order) predicate calculus*, or FOPC), the formulae are propositions that can be augmented with quantified variables, as we will see in Chapter 2. The variables represent values drawn from specified domains.

first-order logic

For most of the rest of this chapter, we will consider variables to be 'illegal' within propositions. As mentioned above, we will be using them within quantified expressions, but that's the next chapter.

second-order logic

First-order logic is all we need to present the material in this book, but logic doesn't stop there. Immediately beyond FOL is *second-order logic* (SOL[3]). SOL extends FOL by permitting variables to represent entire sets, and even functions, rather than just values drawn from sets. There are even higher-order logics if SOL isn't enough for your needs.

An example from programming might help make the FOL – SOL distinction more clear. Most programming languages allow you to (and often insist that you) declare variables before they are used. In C-like languages, you do this with a statement of the form `int a;`. In subsequent statements, a can hold values from the domain of integers. That's also how variables work in FOL. Now imagine that you could declare a variable that can hold entire categories of numbers, rather than individual values. One moment, variable b represents all integers; the next, it represents all rational numbers. That sort of thing is possible with SOL variables.

1.2 Propositional Logic

A formal definition of 'proposition' is a good place to start:

[1] Oooo, foreshadowing!

[2] 'Sentential' from 'sentence' (another substitute for 'proposition'), and 'calculus' meaning any calculation or computation technique (not just the one with derivatives and integration!)

[3] I know, I know; grow up!

proposition

Definition 3: Proposition

A *proposition* (alternatively, *statement*) is a claim that is either true or false within a given context.

We use ‘claim’ in this definition to make it applicable outside of the realm of mathematical expressions.

1.2.1 Simple Propositions

Propositions come in subtypes, of which the simplest is the aptly-named simple proposition.

simple proposition

Definition 4: Simple Proposition

A *simple proposition* is a proposition that includes no logical operators.

Distinguishing propositions from non-propositions can be a bit tricky. A few examples should help.

Example 1:

The following claims are propositions; they are either true or false:

- Radon is a noble gas. (True; one of the six that are natural)
- Bill Gates graduated from Harvard. (False; he dropped out)
- $3 * (4 * 5) = (3 * 4) * 5$ (True; by associativity)
- 9.95 is irrational. (False; it is rational)

These ‘claims’ are not propositions, for the reasons given:

- $2^{\log_8 x} = 4$ (We can’t evaluate its truth without a value for x)
- $a * b = b * a$ (What types are a and b? Does ‘*’ apply to them?)
- Is your refrigerator running? (That’s a question, not a claim.)

- Don't open your present early! (A command, not a claim.)

The first two non-propositions suffer from a lack of context; the variables aren't explained. The last two aren't statements of fact; they do not evaluate to true or false. Please remember that a claim is a proposition when it evaluates to true or false. A proposition is still a proposition whether or not *you* know to which of those it evaluates! Your lack of knowledge does not change the categorization of the statement.

proposition labels

Giving propositions (simple or otherwise) a 'name' aids identification. These names are called *proposition labels* or *statement labels* and are traditionally single lower-case letters written ahead of the proposition and separated with a colon. For example, "r: Radon is a noble gas".

There are two acceptable schemes for choosing proposition labels. The first is to use, in order, the letters 'p', 'q', 'r', 's', etc.[4] The second is to choose a letter that reminds you of the proposition. In the above example, 'r' is the first letter of 'radon'. Yes, we could use longer (more meaningful) names, but soon we will be creating long compound propositions for which short names are a significant convenience.

1.2.2 Compound Propositions

To represent more complex claims, we use compound propositions.

compound proposition

Definition 5: Compound Proposition

A proposition that is a logical combination of simple propositions is a *compound proposition*.

Definition 5 leads to an obvious question: "With what do we combine them?" The answer is, in part, familiar to generations of children who grew up watching the "Schoolhouse Rock" educational videos. ABC's 1973 three-minute grammar animation called "Conjunction Junction", although created to explain a specific part of English sentence structure, is also a nice introduction to two logical operators.[5] Figure 1.1 might bring back some memories.

[4] Why start with 'p'? 'p' is for 'proposition'! It's in every one of those alphabet books for children ... or should be.

[5] You can see this and several others at www.youtube.com/user/SchoolhouseRockVids

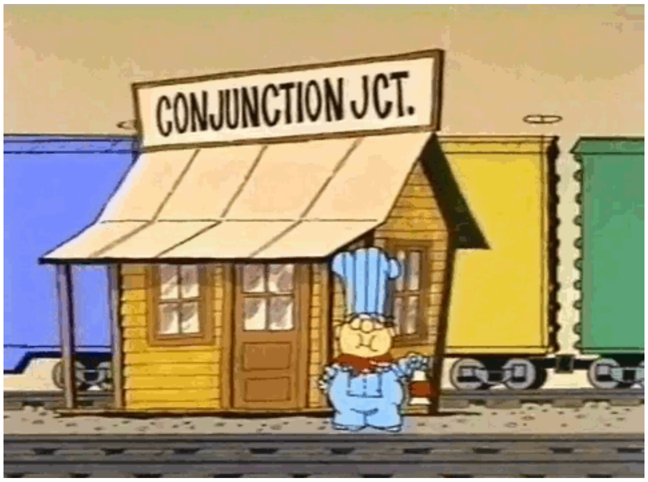

Figure 1.1: ABC's "Conjunction Junction" educational cartoon explained 'and' and 'or' to children using a train car metaphor. Credit: The Walt Disney Company.

1.3 Logical Connectives

This section introduces the commonly-used logical connectives[6] (a.k.a. logical operators) shown in Table 1.

Table 1: Common Logical Connectives

Connective	In Brief	Symbol(s)	LaTeX
conjunction	and	$p \land q$	`\land`
inclusive disjunction	or	$p \lor q$	`\lor`
exclusive disjunction	xor	$p \oplus q$, $p \veebar q$	`\oplus`, `\veebar`
negation	not	$\neg p$, $\overline{p}$	`\neg`, `\overline{}`
implication	if-then	$p \to q$	`\to`
biimplication	iff	$p \leftrightarrow q$	`\leftrightarrow`

Before we get to the connectives, we start with a useful tool for examining the situations in which propositions are true or false: Truth tables.

[6] Negation doesn't do any connecting, as it does not bring together two propositions, but it is a logical operator, and as such is frequently grouped with the others.

1.3.1 Truth Tables

You can think of truth tables as interrogation tools used to get propositions to tell us everything they know. That is, their purpose is to show the complete set of possible evaluations for propositions. Knowing exactly when a proposition evaluates to 'true' is often quite useful.

Table 2: Dissection of a Truth Table

variables ⇒	p	q	$p \vee q$	$q \wedge (p \vee q)$	⇐ *proposition sequence*
	T	T	T	T	
all possible ⇒	T	F	T	F	⇐ *proposition*
truth value	F	T	T	T	*evaluations*
combinations	F	F	F	F	

A truth table is divided into four quadrants, as illustrated in Table 2:

- **Upper-Left**: The variables used by the proposition being evaluated, often ordered to aid the evaluation of the proposition. Note that each variable heads its own column.
- **Lower-Left**: This quadrant lists all of the possible combinations of values that can be assigned to the variables, one combination per row. It is traditional in both logic and mathematics to use the symbols 'T' for 'true' and 'F' for 'false', and to order the rows such that the first row is all 'true', the last is all 'false', and in-between the values change in the same manner as binary digits are incremented. We can look at the pattern another way. The right-most variable's column alternates between 'T' and 'F'. The values of its neighbor to the left alternate also, but in groups of two ('TT', 'FF', 'TT', 'FF', etc.). If there is a third column, we alternate by groups of four (yes, this is based on powers of two).
- **Upper-Right**: Here we use a sequence of propositions to 'build up' to the proposition in question. Typically, each proposition adds an operator to the previous one. As we will demonstrate later, it is sometimes convenient to repeat a column to avoid making silly mistakes.
- **Lower-Right**: This is where we evaluate each proposition in the sequence, left to right, ending with the evaluation of the proposition in

question. Thus, the final (right-most) column is the one in which we are likely the most interested.

Categories of Propositions

The following definitions provide three categories of propositions, based on how often they evaluate to true and false.

tautology

Definition 6: Tautology

A *tautology* is a proposition that always evaluates to true.

contradiction

Definition 7: Contradiction

A proposition that always evaluates to false is a *contradiction*.

contingency

Definition 8: Contingency

A proposition that is neither a tautology nor a contradiction is known as a *contingency*.

Example 2:

Problem: Is the compound proposition $q \wedge (p \vee q)$ a tautology, a contradiction, or a contingency?

Solution: It's a contingency, as Table 2 shows. The column of boolean values beneath the proposition includes both true and false, which makes $q \wedge (p \vee q)$ a contingency.

Truth tables are excellent tools for categorizing propositions, because they show all possible evaluations. Tautologies and contradictions can also be determined by applications of logical equivalences; keep reading!

1.3.2 Conjunction

Consider this English sentence:

> *Amit bought a tablet and installed a chess app.*

conjunction

This sentence contains two separate propositions (*Amit bought a tablet* and *[he] installed a chess app*). We know that these are propositions because they are either true or false — Amit either bought a tablet or he did not, and he did or did not install a chess app. The 'and' between them makes the complete sentence a *conjunction*, ($p \land q$, LaTeX: `\land`) which is the type of compound proposition that is true only when both participating propositions are true. The truth table for conjunction can be found in Table 3, along with those of the other common connectives.

Table 3: Truth Tables of the Common Logical Connectives

				inclusive	*exclusive*		
		negation	*conjunction*	*disjunction*	*disjunction*	*implication*	*biimplication*
p	q	$\neg q$	$p \land q$	$p \lor q$	$p \oplus q$	$p \to q$	$p \leftrightarrow q$
T	T	F	T	T	F	T	T
T	F	T	F	T	T	F	F
F	T	-	F	T	T	T	F
F	F	-	F	F	F	T	T

Turning an English sentence into the logical notation shown in Table 3 is a four-step process: (1) Identify the simple propositions, (2) assign proposition labels to them, (3) identify the logical operators, and (4) make all of the substitutions. Example 3 demonstrates this process.

Example 3:

Problem: Convert to logical notation this sentence: *Bobby found, bought, and wrapped the present.*

Solution: As with the first example, this sentence is missing some parts. As English speakers, we are used to this, and can insert the missing parts if necessary. We will do that here to make the conversion steps plain: *Bobby found the present and Bobby bought the present and Bobby wrapped the present.* This is awkward but correctly-structured English.

Step 1: Identify the simple propositions. Having expanded the original sentence, the three simple propositions are easy to see: *Bobby found the present*, *Bobby bought the present*, and *Bobby wrapped the present.*

Step 2: Assign proposition labels. To select meaningful labels, look at the subjects, verb, and objects of the propositions. Here, the subjects and objects ('Bobby' and 'present') are the same in all three, but the verbs all start with different letters, making our label choices easy:

f :	Bobby found the present
b :	Bobby bought the present
w :	Bobby wrapped the present

Step 3: Identify the logical operators. This step is not much of a challenge, as we have covered only one: conjunction. With three simple propositions, we need two conjunctions to tie them together, and we have them.

Step 4: Make all of the substitutions. Replacing the English propositions with our labels and using '$\wedge$' for 'and' produces the answer.

The logical notation version of *Bobby found, bought, and wrapped the present* is $f \wedge b \wedge w$, using the labels defined above.

1.3.3 Inclusive and Exclusive Disjunction

Consider this compound proposition:

He won the game by defeating the boss or by collecting all of the relics.

What are the circumstances under which this claim could be true? If he defeated the boss, but missed a relic, we would consider this to be true, because the connective ('or') allows for either one to be true. Similarly, if he found all of the relics but let the boss win, the claim is still true because, again, one part of it is true. If neither action took place, we would consider the statement to be false – there were two ways to win, and he didn't achieve either of them.

inclusive disjunction
exclusive disjunction

That leaves one option: What if both parts are true? If the ‘or’ is an *inclusive disjunction*, ($p \vee q$, LaTeX: `\lor`) then yes, we consider the claim to be true. But if the ‘or’ is an *exclusive disjunction*, , ($p \oplus q$, LaTeX: `\oplus`) the claim is considered to be false. The names provide an easy way to remember this distinction: inclusive disjunction includes the case where both parts are true, but exclusive disjunction excludes that case. Table 3 shows that this is the only difference between these two types of disjunction.

Unfortunately, in English the ‘or’ by itself does not tell the reader which interpretation is correct. Sometimes, we can tell by context. In other situations, adding a few extra words makes the difference.

Example 4:

Adding “or both” to the end of a disjunction tells the reader to assume inclusive disjunction: *You can type your ID number or your full name, or both.* Similarly, “but not both” means that exclusive disjunction is intended. Prefixing the statement with “either” gives a hint of exclusivity, but still leaves the reader in some doubt: *Either rake the yard this afternoon or this evening.* Can the raker do half of the yard in the afternoon and the other half in the evening? Even with the addition of ‘either’, it’s hard to tell. Ah, the joys of natural language! Sometimes, we have to ask for additional clarification to get the meaning correct.

Example 5:

Problem: Determine the types of disjunction used in each of these sentences:

(a) Ramone or Elias scored the winning hockey goal.
(b) Fill in the circles with pen or pencil.
(c) The meal includes mashed or french-fried potatoes.

Solution: (a) is exclusive; the scorekeeper must credit the goal to just one player. (b) is inclusive; as the type of mark does not matter, a combination is acceptable. (c) is likely intended to be exclusive, though it is possible the waiter could arrange for you to receive some of each variety.

Example 6:

Problem: Why is Wei late? Express the answer *Wei overslept or missed the 8 a.m. bus* in logic.[7]

Solution: Using the four-step process given above (see page 8 and demonstrated in Example 3):

1. The simple propositions are *Wei overslept* and *Wei missed the 8 a.m. bus.*
2. o: Wei overslept m: Wei missed the 8 a.m. bus
3. It's possible that Wei was late for both reasons, making inclusive OR the best disjunction option here.
4. In logic: $o \vee m$.

This is a good time for a word to the wise: *Do not 'over-think' the meanings of propositions*, especially not in this book! When people learn about converting from English to logic, they sometimes let their imaginations get the best of them, and start inventing interpretations that lead to indecision and ultimately an indefensible answer.

Example 7:

Problem: True or False: The sky is blue with one bright light source or black with lots of them.

'Solution': "Hmmmm. Sounds like an exclusive 'or'; it is either daylight (blue sky) or night (black). But there are two transition periods, with the sky moving between blue and black. Then again, at different places on Earth, the sky can be blue and black at the same time, which is inclusive. And is the moon a light source? Sure, it only reflects sunlight, but it does reflect it well enough to be seen when the sky is blue, so that

[7] Might as well; Wei's not here yet, so you've got time to kill.

Figure 1.2: "Need to type up your discrete structures homeworks? Feel free to use any typeface, point size, and symbols you wish . . . so long as this keyboard[8] has 'em." Credit: Wikimedia Commons

could be a second light source. Wait – if we were on the moon, the sky is always black; it cannot be blue at all! And what about alien planets with sulphuric atmospheres, perpetual clouds and two suns? Boy, what a stupid question . . . I'll just say 'Yes'."

Solution: An answer of 'Yes' is unacceptable when the question asks for 'true' or 'false'! Go with the most straight-forward interpretation: True, by exclusive 'or'. We are not trying to trick anyone with our questions.

1.3.4 Negation

unary connective

binary connective

As shown in Table 3, negation takes just one operand, making its truth table half the size of the others. It is a *unary connective*, unlike conjunction and disjunction, which take two operands and are therefore termed *binary connectives.*

Over time, several different notations for negation have been used. In the days of typewriters, the set of available symbols was quite limited (see Figure 1.2), and as a result most of them had multiple meanings. Combine that with the limits on the exchange of ideas, and it is not hard to see how variety would flourish. In the case of negation, the modern notations are the

[8] This is a picture of an old European QWERTZ typewriter keyboard. Yes; QWERTZ, not QWERTY. Seriously!

negation symbol ($\neg p$, LaTeX: `\neg`) and the *overline* or *overbar* ($\overline{p}$, LaTeX : `\overline{}`). Older negation notations that are still used occasionally include $\sim p$ and p'.[9]

Example 8:

Problem: Use a truth table to show that $(p \vee q) \vee \overline{p}$ is a tautology.

Solution: Recall that a tautology always evaluates to true. We can use a truth table to show that, regardless of the combinations of true and false that are assigned to p and q, the evaluation of that expression will be true.

p	q	$p \vee q$	$\overline{p}$	$(p \vee q) \vee \overline{p}$
T	T	T	F	T
T	F	T	F	T
F	T	T	T	T
F	F	F	T	T

Occasionally, students go a little crazy when they negate propositions. Example 9 demonstrates both right and wrong ways to do it, and Example 10 shows that not all propositions have easily-constructed negations.

Example 9:

Problem: Let m label the proposition *The mountain is tall.* In English, express $\overline{m}$.

Correct Solution: *The mountain is not tall.* The safest way to express the opposite of 'tall' is by saying 'not tall'.

A Probably Acceptable Solution: *The mountain is short.* 'Short' is usually considered to be an antonym of 'tall', making this an acceptable negation in most situations. Trouble occurs when you start asking annoying questions such as, "What about ordinary, average-height mountains?"

[9] $\sim$ is called a *tilde* (LaTeX: `\sim`)

Incorrect Solution: *The valley is tall.* The characteristic of mountains known as 'height' is what we need to negate. Changing the subject does not address the problem.

Another Incorrect Solution: *The mountain was tall.* Changing the verb tense does not address the problem, either.

Example 10:

Problem: Negate the proposition *Education funding has decreased and test scores are down.*

Solution: Let p represent the first simple proposition and q the second. We know that the truth table of $p \wedge q$ reads T, F, F, F from top to bottom (see Table 1 if you need to refresh). The negation $[\neg(p \wedge q)]$ needs to evaluate to F, T, T, T to be a complete negation.

If we try *Education funding has increased and test scores are up* as the negation, we have two problems. First, is 'increased' the negation of 'decreased'? How does 'staying the same' fit in? (The same can be said for 'up' and 'down'.) We will take our earlier advice and not overthink these. Second, the truth table column for this version $[\neg p \wedge \neg q]$ is F, F, F, T instead of the needed F, T, T, T.

As we will learn in Section 1.5.2, a pair of logical equivalences known as De Morgan's Laws tell us that the correct negation (still ignoring the annoying 'stay the same' possibility) is: *Education funding has increased or test scores are up* $[\neg p \vee \neg q]$.

A classic way to express negation in English is to prefix the statement with "It is not the case that". If we try that on the proposition from Example 10, we produce: *It is not the case that education funding has decreased and test scores are down*, but what is being negated is not clear. Are we negating just the first simple proposition or the entire compound proposition? Natural languages are notoriously imprecise.

Something else to remember: It is very easy to overlook an overbar, particularly one that covers an entire compound proposition. Be sure to watch for them!

1.3.5 A Digression: Well-Formed Formulae

It is likely that you have typed a mathematical expression into an assignment statement of a program, attempted to compile it, and had the compiler complain that your expression is malformed. Chances are, the compiler complained because your expression was not a correctly structured. A *well-formed formula* (frequently abbreviated '*wff*') is a grammatically-correct expression of the intended language. Most often, the term is used in mathematics, logic, and programming, but it applies to any language usage, even the construction of English sentences.

well-formed formula

Example 11:

Consider this assignment statement, which follows the syntax of a variety of programming languages:

```
      result = 8 + * 9;
```

`8 + * 9` is not a well-formed formula.[10] Multiplication of integers is a binary operation, and in a language that uses infix notation (that is, nearly all of them), the proper form is *operand * operand*. The 9 is the second operand of the multiplication operator, but the first is missing.

Compound logical propositions are statements of a language, just as arithmetic expressions are. Fortunately, the grammatical rules of the language of logic are much the same as those of arithmetic and algebra. Briefly, assuming that p and q are propositions, then based on the logical operators we've seen so far:

1. If p and q are propositions, then they are wffs.
2. If r and s are wffs, then so are (r), $\neg r$, $r \wedge s$, $r \vee s$, and $r \oplus s$.

[10] However, `foo = 8 * + 9;` is usually considered to be well-formed. If you don't see why, put it into a program and look at the value assigned to `foo`.

Example 12:

Problem: Show that $\overline{t \oplus u}$ is a wff in logic, given that t and u are propositions.

Solution: By (1), above, t and u are wffs. By (2), $t \oplus u$ is a wff. And, also by (2), so is $\overline{t \oplus u}$.

Example 12's demonstration shows the foundation of the process used by programming language compilers to check that a program is a correct expression of that language.

1.3.6 Implication (a.k.a. Conditional Proposition)

Programming languages provide selection statements that allow computers to execute one or more commands based on the evaluation of a condition. In most languages, the form is something like `if <condition> <action(s)>`, which mimics one of the forms conditionals take in English. The interpretation is much like the English version as well: The action(s) are performed only when the condition is true, and ignored when it is false.

Take this English sentence:[11]

If you start the car, then the engine will run.

implication

We can convert this compound proposition to logic as we did with our previous examples. Here we again have two (simple) propositions (s*: you start the car* and e*: the engine will run*). The symbol for *implication* is a right-pointing arrow ($s \to e$, LaTeX: `\to`). Rewriting this is easy; understanding how to interpret the logic is harder.

Assume that both s and e are true; that is, you do start the car, and as a result the engine does run. You would accept that $s \to e$ is true. Now consider that the engine does not run (e is false) even though you started the car. You would have no trouble accepting that the sentence is false. But how do we interpret the truth of the statement when s is false? Certainly, we do not expect the engine to spring to life all by itself.[12] More to the point, is the entire *if you start the car, then the engine will run* statement a true statement when we do not have a chance to see it in action? The answer

[11] ...please! (`en.wikipedia.org/wiki/Henny_Youngman`)

[12] Outside of cartoons and horror movies, that is.

is, perhaps surprisingly, ‘yes’! We have no justification for saying that the statement is false, so we accept that it is true. This assumption of truth is known as *vacuous truth.*

vacuous truth

Definition 9: Vacuous Truth

Given a statement that can be expressed as an implication $p \to q$, the statement is *vacuously true* when p is false.

In English, ‘vacuous’ is synonymous with ‘empty,’ explaining why vacuous truths are sometimes known as empty truths.

Example 13:

Question: Is the statement “If Godzilla is an astronaut, he is wearing blue ear plugs” vacuously true?

Answer: Godzilla is a fictional monster. As such, the set of astronaut Godzillas is empty. Because “Godzilla is an astronaut” is false, and because **F** → anything is true, yes, the statement is vacuously true.

You could just memorize the implication truth table (which can be found in Table 3), but looking at `if` statements in programming languages provides another perspective on this interpretation. When you write a program and the computer translates it (to machine language or bytecode or whatever), the translator verifies that the structure of the code meets the specifications of the language. That is, your `if` statement is deemed to be ‘correct’ or ‘true’, even though you have yet to execute the program. Similarly, in English, you can check that the sentence syntax is correct without knowing the meaning of all of the words in it, so long as you know their roles (noun, verb, etc.).

The ‘if’ part and the ‘then’ part of implications go by a variety of pairs of names, as shown in Table 4:

Table 4: Names of p and q in “if p, then q”

	p	—	q
(a)	hypothesis	—	conclusion
(b)	antecedent	—	consequent
(c)	sufficient	—	necessary

Avoid the temptation to "mix and match" the terms in Table 4. That is, do not pair 'sufficient' with 'consequent', or people might wonder about your education.

English, being a language with much flexibility, supports many ways of writing "if p, then q". Table 5 covers nineteen (including the 'when' and 'whenever' substitutions), but even so it is by no means an exhaustive list.

Table 5: Some Other Ways of Saying "if p, then q"

(a) if p, q
(b) p implies q
(c) p infers q
(d) p only if q
(e) p is sufficient for q
(f) q if p
(g) q is implied by p
(h) q follows from p
(i) q unless $\overline{p}$
(j) q is necessary for p

NOTE: You can replace the word `if` with `when` or `whenever` without changing the logical interpretation.

Example 14:

Problem: Express $(\overline{o} \wedge c) \rightarrow a$ in conversational English using the *q is sufficient for p* form, given that:

o: Wei oversleeps
c: Wei catches the 8 a.m. bus
a: Wei attends the meeting

Solution: The negation of *Wei oversleeps* is *Wei does not oversleep.* Dropping the rest of the propositions in place, adding the operators, using the requested form, and making tweaks to the English to make it read (reasonably) conversationally, we get: *Wei not oversleeping and catching the 8 a.m. bus is sufficient for him to attend the meeting.*

Let's face it: It's really hard to make sentences that use *is sufficient for* and *is necessary for* beautifully conversational!

Example 15:

Problem: Rewrite each of these English expressions of implication in "if *antecedent* then *consequent*" form:

(a) You'll hurt your back if you lift that box.
(b) The light comes on only if the circuit is not open.
(c) An installed battery is necessary for the remote control to work.

Solution: Part (a) is straight-forward; it matches the "*consequent* if *antecedent*" form. In if-then form: *If you lift that box, then you'll hurt your back.*

Part (b) is a bit more complex. There are four forms in Table 5 that include the word "if". In just one of those does the hypothesis not immediately follow the "if", and that is the "only if" case. Be mindful of that word "only"! With that in mind, the hypothesis and conclusion are already in the proper orientation for an if-then: *If the light comes on, then the circuit is not open.* Notice that the "not" just stays where it is; compound propositions can serve as hypotheses and conclusions, too.

Part (c) demonstrates that just because you can use a particular form does not mean that you should. Outside of nerdy sitcoms and logic classes, very few people talk like that (that is, the form is not very conversational). Happily, we can convert it to something more reasonable: *If the remote control works, then a battery is installed.* Do not worry about matching verb tense and word arrangements within the component propositions, so long as the result has the same basic meaning and is correctly-structured English.

"Only if" sentences, such as part (b) of Example 15, often raise a question: What is wrong with putting the pieces in the other order? In this case, that would produce the sentence *if the circuit is closed, then the light comes on*, which sounds pretty reasonable. But is anything else needed for the light to come on? Sure; for starters, the filament (or LED or . . .) must not be broken and the power source must supply adequate power. That is, the circuit being closed is not sufficient by itself for the light to come on. The given answer (*if the light comes on, then the circuit is closed*) is logical; given that the light came on, we can reasonably conclude that the circuit was closed.

People sometimes play on the flawed understanding some people have of conditionals to influence their opinions. The practice of "push" polling in politics is a classic example.

Example 16:

Ahead of the 2000 U.S. presidential primary in South Carolina, a telephone poll (apparently authorized by the George W. Bush campaign team) asked this question: *Would you be more likely or less likely to vote for John McCain for president if you knew he had fathered an illegitimate black child?* The poll was designed to suggest to ill-informed voters that McCain's adopted Bangladeshi daughter, Bridget, was biologically his. Notice the 'if' in the question. There is no claim that such an act occurred, but listener susceptible to suggestion could be left with that impression. You know better: First, you know that the truth of a conditional depends on the true of the antecedent, and second, you know that questions are not propositions.

This survey technique is known as "push" polling because it attempts to sway ('push') the listener to a particular point of view. Signatories of the American Association of Political Consultant's Code of Ethics are prohibited from using "false or misleading attacks".

Converse, Inverse, and Contrapositive

Three adjustments to the basic $p \to q$ implication have enough utility to be worth special mention.

converse

- The *converse* of $p \to q$ is the implication $q \to p$. To form the converse, just swap the hypothesis and the conclusion.

inverse

- The *inverse* of $p \to q$ is the same implication with both the antecedent and the consequent negated. That is, the inverse of $p \to q$ is $\overline{p} \to \overline{q}$.

- Finally, the *contrapositive* is the original implication with both converse and inverse applied to it – the hypothesis and conclusion are each negated and swap positions. Symbolically, the contrapositive of $p \to q$ is $\overline{q} \to \overline{p}$. *contrapositive*

Many people find inverse and converse to be hard to keep straight. Here is a mental trick that might help: *In*verse *in*cludes negation. Because contrapositive includes both adjustments, it is easy to distinguish from the other two.

Why are these worth knowing? Apart from jokes about athletic footwear, archaic ways of describing mating, and bad puns,[13] they are useful examples when discussing logical equivalence, as we will see in Examples 24 and 32.

1.3.7 Biimplication (a.k.a. Biconditional Proposition)

Does this sentence sound pretentious to you?

I will eat truffles if and only if they are in a French wine reduction.

The topic certainly raises the pretentiousness level, but what about that "if and only if" part? Why not just say "if" once instead of reiterating it?

The answer is that "if and only if" actually is not repeating the "if". Rather, it is meant to be taken literally: It is a combination of the "if" and the "only if" expressions of implication. We can see the logical interpretation by breaking down the example. Assume that t labels *I will eat truffles* and w labels *[the truffles] are in a French wine reduction.*

if	**and**	**only if**	
t if w		t only if w	← as separate propositions
if w, then t		if t, then w	← in if-then form
$(w \to t)$	$\wedge$	$(t \to w)$	← converted to logic notation

Now the meaning is (hopefully!) clear: "if and only if" combines two implications into one statement, providing a compact way of writing this form of compound proposition. It may sound pretentious, but it is really just a shortcut.[14] This logical operator is known as a *biconditional* or a *biimplication*, *biconditional*

[13] Converse is an athletic apparel company, and a rarely-used word to describe sex. ("No worries, Mom; we . . . conversed last night, that's all. It was logic homework, I swear!") As for puns, how about "A negative poet writes inverse"?

[14] Now you can be pretentious, educated, and lazy all at the same time!

biimplication

and is represented with the symbol ↔ (LaTeX: `\leftrightarrow`). Another way of writing *if and only if* in English is with the abbreviation *iff*,[15] which you may have seen in other mathematics classes because that is where it is most often used.[16]

Table 3 (all the way back on page 8) shows that biimplication is true only when the operands match (sometimes stated as "have the same parity"). That is, a biconditional is true when its arguments are both true or when they are both false. Remember that you do not have to rely on your memory of this fact; if need be, you can reconstruct the above derivation and then create the truth table for $(p \rightarrow q) \wedge (q \rightarrow p)$.

People frequently use a conditional expression when a biconditional would correctly give more useful information, as Example 17 demonstrates.

Example 17:

Problem: Which is more helpful: $x = 2y$ *if* $y = x/2$, or $x = 2y$ *iff* $y = x/2$, when x and y are integers?

Solution: First, both are correct; you can safely state either one. The difference is that the first version is just half of the second; that is, the second includes the first, and adds *if* $x = 2y$*, then* $y = x/2$. As both parts are correct over the integers, the second version provides more information and it therefore more helpful.

1.3.8 Precedence and Associativity of Logical Connectives

precedence
associativity

You are likely familiar with the ideas of operator precedence and associativity from mathematics and the programming languages you have learned. *Precedence* tells us which of a group of operators to perform first in an expression when no parentheses are provided, and *associativity* provides an evaluation order when the operators have the same precedence.

Most discrete structures books provide a precedence table for the logical operators. They agree on the rough structure (negation has highest precedence, followed by conjunction and disjunction, with the forms of implication

[15] Speaking of pretentious: See how the two f's in 'iff' are joined? In typography, that's called a *ligature*. LaTeX uses them automatically, but that behavior can be overridden.

[16] In LaTeX , `\iff` produces the symbol $\iff$ instead of ↔. Some people like the double-line arrows for implication and biimplication. We like the simpler single-line arrows.

next), but they often disagree on the smaller details (e.g., does conjunction have higher precedence than disjunction, or are they the same?). Instead of adding to the confusion, for this book we adopt a simple rule: *Use parentheses to avoid confusion.*

Associativity is much less troublesome: *All of the binary logical connectives are left-associative.* That is, assuming that $\diamond$ is a left-associative binary logical connective, when evaluating the expression $a \diamond b \diamond c$, $a \diamond b$ (the left-most operator) is evaluated first. To force right-associativity, add parentheses, as in $a \diamond (b \diamond c)$. Negation is naturally right-associative. Given the expression $\neg\neg p$, the only way it can be evaluated is to perform the right-most negation first; if we try to make it left-associative, we would be using a negation operator as the operand to the other negation operator.

1.3.9 A Digression: Logical Bit-Operators in Programming Languages

So-called "systems" programming languages (that is, languages that are designed to create programs that interface with hardware more so than with people) include operators that consider individual bits to be their operands. Of course, most computers work with memory in larger 'chunks' (words) than individual bits, but these operators still apply to the component bits of those words.

The C family of programming languages (which includes C++ and Java) provides several bit-level operators. Due to the fact that bits have just two possible values (0 and 1, or 'off' and 'on', or 'false' and 'true'), these operators include negation, conjunction, and disjunction.[17]

Table 6 lists and demonstrates four of the logical operators as they appear in C as bit-level operators, along with two bit-shifting operators as examples of additional bit-level operators. The right-most column shows most clearly how the corresponding pairs of bits from the operands are processed by the binary operators.

Table 6: Logical Bit Operators in C-family Languages

[17] As we will soon see, implication is a composite operator – we can built it from disjunction and negation – and so programming languages do not need to provide it as a separate operator.

Operator	Name	Example (Dec.)	Example (Binary)
$\sim$	Complement	$\sim 12 = -13$	$\sim 00001100 = 11110011$
&	AND	$12 \,\&\, 10 = 8$	1100 & 1010 1000
$\mid$	OR	$12 \mid 10 = 14$	1100 $\mid$ 1010 1110
$\wedge$	XOR	$12 \wedge 10 = 6$	1100 $\wedge$ 1010 0110
>>	Shift Right	$33 >> 1 = 16$	$00100001 >> 1 = 00010000$
<<	Shift Left	$33 << 2 = 132$	$00100001 << 2 = 10000100$

Those of you who have programmed with a C-family language may have wondered why logical AND, for example, is represented by a pair of ampersands rather than a just one. Table 6 answers the question: The single-ampersand version is used for bit-level conjunction, and the double-ampersand version is used for variable-level conjunction. If you are wondering why $\sim 12 = -13$ instead of -12 and how shifting bits can be useful, those topics are often covered in computer architecture classes.[18]

Knowledge of bit-wise logical operators is useful in some interesting places. Example 18 introduces one such situation.

Example 18:

UNIX and UNIX-like operating systems maintain a 9-bit default permission value (frequently `110 110 110`, or 666_8) and each process maintains another bit string called the *umask*. Together, these bit patterns are used to give new files created by the process initial access permissions. Here's how.

Say that the default permission string is as stated above and the umask value is 037_8 (= 000 011 111 in binary). The result of the bit-wise AND-ing of the complement (a.k.a. negation) of the umask to the default

[18] Too much work? Fine; be that way: Computers usually use a representation called *two's complement* for signed integers, and shifting is an efficient way to multiply and divide by powers of two. You and http://www.wikipedia.com can take it from there.

permission string is the permission string assigned to newly-created files. In this case:

	`000`	`011`	`111`	[umask]
	`111`	`100`	`000`	[complement of umask]
&	`110`	`110`	`110`	[default permissions]
	`110`	`100`	`000`	[the file's permissions]
	`rw-`	`r--`	`---`	[as reported by `ls -l`]

What does this mean? The new file can be read or written by the file's owner, can be read (but not written to) by users with group access, and is inaccessible to anyone else. Users can adjust their command shell's umask. Now you know how to chose a suitable value.

1.4 More English ⇔ Logic Conversions

Remember that we are studying the basics of logic to prepare for proofs. Statements that we wish to prove correct are sometimes expressed in logic or mathematical notation for us, but they can also be expressions in natural languages (like English). Before we can apply logical principles to the statements, we have to convert them to logic to make them easier to work with and to solidify their meanings. Converting from logic to a natural language is an equally-important skill. It is good to be able to produce a logical result, but it is even better to be able to explain the meaning of the result to those not versed in logical notation.

In section 1.3.2 (see Example 3 in particular), we showed how to convert from English to logic using a four-step process: (1) Identify the simple propositions, (2) assign proposition labels to them, (3) identify the logical operators, and (4) make all of the substitutions. Examples 19 and 20 further demonstrate the challenges of English to logic conversions.

Example 19:

Problem: Express this sentence in logic notation: *He loses the election if he loses Ohio or Virginia.*

Solution: Steps 1 and 2 identify the component (simple) propositions and label them. When negations are involved, think positively. That is, state the propositions without negations. (Why? Because negations are operators, and that's Step 3.) Following this philosophy, we have three simple propositions – e: *He wins the election*, o: *He wins Ohio*, and v: *He wins Virginia.*

Now for the operators: We clearly have an implication (the 'if' in the middle) and a disjunction (the 'or' between the states) – but which variety of disjunction is it? Presumably, if the candidate loses both states he will lose the election by an even wider margin, making inclusive disjunction the reasonable choice. Another question: How many negations do we have, two or three? One applies to e, but is there just one covering the disjunction of o and v, or is there one negation for each proposition? We can answer this by expanding the antecedent from *he loses Ohio or Virginia* to *he loses Ohio or he loses Virginia.* Three negations it is.

At last, step 4 produces the final answer: $(\neg o \vee \neg v) \to \neg e$.

You may be wondering if the 'one negation vs. two' issue even matters. It does, as a truth table would make clear. $\overline{o} \vee \overline{v}$ and $\overline{o \vee v}$ differ when o and v have different values.

Example 20:

Problem: Express this sentence is logic notation: *The sum of two odd integers is also an odd integer.*

Solution: This problem is challenging for two reasons: First, it deals with math, which we have yet to try to express in logic. Second, it is missing all of our logic keywords! This is the sort of sentence we will see when we discuss proofs.

To express this sentence in logic, we need to rewrite it so that the propositions and logical connectives are clear. When working with math, it can help to add variables. In this example, we can introduce two variables (i and j will do) to represent the odd integers being added. This means that our propositions are p: $i \in \mathbb{Z}^{odd}$, q: $j \in \mathbb{Z}^{odd}$, and r: $(i + j) \in \mathbb{Z}^{odd}$.

Now to connect them. The implicit assumption is that both i and j are provided to us; given them, we add them and claim that the result is odd. That is, we are assuming that they are odd integers – *if* they are both odd, *then* their sum is odd. 'Both' suggests conjunction, and the if-then is implication. Thus, we can rewrite the original sentence: *If i and j are odd integers, then $i + j$ is an odd integer.* In this form, and having already identified the propositions and connectives, the conversion to logic is straight-forward: $(p \wedge q) \rightarrow r$.

You may be wondering if $p \wedge q \wedge r$ is an acceptable alternative to $(p \wedge q) \rightarrow r$ in Example 20. It is not. The sum is a result of a process (here, addition), not something that is also provided to us. As there is a conclusion to be reached, implication best fits this situation.

When converting from logic to English, the trick is to make the result conversational; that is, to write the English version so that it sounds as though it were written for a person by a person, rather than by a mechanical conversion process, while retaining the correct meaning. Example 21 should help make this distinction clear.

Example 21:

Problem: Express this logical expression in conversational English: $\overline{\overline{i} \wedge \overline{u}}$, where i: *education funding has increased* and u: *test scores are up*

Solution: You may have noticed that this is a slightly-restructured version of Example 10. In particular, notice that there is an overline over the entire expression. Expressing in conversational English the negations of i and u is easy if we aren't too picky, as we have already seen in that earlier example: $\overline{i}$: *education funding has decreased* and $\overline{u}$: *test scores are down.*

Expressing the negation of the conjunction is more challenging. The prefix phrase *it is not the case that* is the lazy (and not very conversational) solution for expressing a negated proposition. Using that here produces a difficult-to-interpret English version: *It is not the case that education funding has decreased and test scores are down.* Does the prefix apply to both parts or just the first part? The lack of a comma ahead of the 'and' suggests both, but still leaves doubt. We can create a more clear version by being a little more creative: *That both education funding has decreased and test scores are down is not true.* Adding 'both' and moving the negation to the end makes the meaning more plain – and acceptably conversational.[19]

To repeat what Example 10 said: We will see another way to express this situation when we cover logical equivalences. In the meantime, we hope you appreciate why we want to express ideas in logic rather than in English. Natural languages are not famous for their precision.

Example 22:

Problem: Express this logical expression in conversational English: $(p \wedge q) \rightarrow r$, where p: $i \in \mathbb{Z}^{\mathrm{odd}}$, q: $j \in \mathbb{Z}^{\mathrm{odd}}$ and r: $(i + j) \in \mathbb{Z}^{\mathrm{odd}}$.

Solution: Yes, this is just Example 20 in reverse. We are not merely lazy; we are also doing this to make a point.

Translating the logic directly to English is possible but produces a result that is unsatisfying: *If i is an odd integer and j is an odd integer, then $i + j$ is an odd integer.* It is difficult to complain about the correctness of the English version, but we can make it less awkward – more conversational – by reducing the repetition: *If i and j are both odd integers, then $i + j$ is also odd.* And, for those who feel that a sentence is not English if it contains variables and math expressions, we can use the version that started Example 20: *The sum of two odd integers is also an odd integer.*

What was our point? We have two, actually: First, converting directly to English is likely to be unsatisfactory if your goal is conversationality, and second, there are different levels of informal expression – what one person feels is

[19] Want it to be more conversational? Imagine a political candidate's debate: "My knuckle-dragging, loose-jawed and lightly-educated opponent's claim that both education funding has decreased and test scores are down has as much truth as Hollywood movie accounting . . . but fewer mysterious explosions."

conversational may not be acceptable to another. Unfortunately, distinguishing the two perspectives is likely to be difficult.

1.5 Logical Equivalence

1.5.1 Two Definitions of Logical Equivalence

As there are multiple ways of saying the same thing in English, there are multiple ways to express logical propositions so that they all produce the same column in a truth table. This idea is the core of our first definition of logical equivalence.

logical equivalence

Definition 10: Logical Equivalence

Two propositions p and q are *logically equivalent* (written $p \equiv q$, LaTeX: `\equiv`) when both evaluate to the same result when presented with the same input.

Example 23:

Back in section 1.3.1, we used a truth table in Table 2, but only for labeling its sections. Here is the unadorned table:

p	q	$p \vee q$	$q \wedge (p \vee q)$
T	T	T	T
T	F	T	F
F	T	T	T
F	F	F	F

You may not have noticed at the time, but the columns for q and $q \wedge (p \vee q)$ are identical. As the truth table covers all possible inputs to these two propositions, and as their evaluations match on all of those inputs, $q \equiv q \wedge (p \vee q)$.

Example 24:

Problem: Show that $r \rightarrow s \;\equiv\; \neg s \rightarrow \neg r$ and that the converse of $r \rightarrow s$ is equivalent to its inverse.

Solution: A single truth table can that show both of these claims are, perhaps surprising, correct:

		(Original)	(Converse)	(Inverse)	(Contrapositive)
r	s	$r \rightarrow s$	$s \rightarrow r$	$\neg r \rightarrow \neg s$	$\neg s \rightarrow \neg r$
T	T	T	T	T	T
T	F	F	T	T	F
F	T	T	F	F	T
F	F	T	T	T	T

It's important to remember that, while the inverse and the converse are equivalent to each other, neither is equivalent to the original implication. Only the contrapositive is equivalent to the original.

This is a good time to clear up a common point of confusion: *In logic, equality is not the same as equivalence.* q and $q \wedge (p \vee q)$ are equivalent, as Example 23 shows, but they are not equal because they are different expressions (one is a simple proposition and the other is compound). Later, we will further distinguish equality and equivalence by introducing the idea of *equivalence relations.*

You may be tempted to use this to argue that, in algebra, "$a+b$" and "$b+a$" should be equivalent but not equal. Restate this is logic (that is, change "+" to "$\wedge$" and the domain from real to boolean) and we would agree with you. Logic and algebra share common characteristics (such as commutativity), but they are not the same field of study.

Example 25:

As a computer programmer, no doubt you have found yourself, probably very late at night, seemingly unable to construct a condition for an `if` statement that will make your program run correctly. The more desperate you became, the wilder your modifications became. Eventually, you stumbled on a condition that worked, and you never looked at that code again.

If you had, you might have discovered a really good logical equivalence example! Imagine that this pseudocode **if** statement holds your hard-found condition:

```
    if (temp > tmp or tmp2 = temp) and tmp < temp then ...
```

(You really need to start using more meaningful variable names, by the way.) This compound proposition – yes, that's what it is – includes the same condition twice; `tmp < temp` is the same as `temp > tmp`. With the application of commutativity on both the **and** and **or**, this turns out to be . . . the proposition $q \wedge (p \vee q)$ from Example 23! Because that is known to be equivalent to just q, we could replace the original **if** statement with:

```
    if tmp < temp then ...
```

and make our code both more efficient (fewer operators to evaluate) and easier to understand (especially with better variable names).

True, back when we defined propositions (section 1.2), we said that expressions with variables were not acceptable because we did not know what the variables represented. In a correctly-structured computer program, when a condition is evaluated, the variables all have values. If we imagine replacing the variables with their values, it is easy to see why a condition in a program is actually a proposition.

Recall that biimplication is true only when the operands have the same value. (Or, refresh your memory with a look back at Table 3.) This fact leads to an alternative definition of logical equivalence.

logical equivalence

Definition 11: Logical Equivalence

Propositions p and q are *logically equivalent* (written $p \equiv q$, LaTeX : `\equiv`) if $p \leftrightarrow q$ is a tautology.

Example 26:

Here is the truth table from Example 23, augmented with an additional column:

p	q	$p \vee q$	$q \wedge (p \vee q)$	$q \leftrightarrow [q \wedge (p \vee q)]$
T	T	T	T	T
T	F	T	F	T
F	T	T	T	T
F	F	F	F	T

Because $q \equiv q \wedge (p \vee q)$, their results will always match, their biimplication will always be true, and thus it is a tautology.

1.5.2 Common Logical Equivalences

An infinite number of logical equivalences can be created. A few dozen of those turn out to be particularly useful in problem-solving. Listed below is our collection, grouped in tables based on the logical connectives they use. Names are provided for many but not all. Be aware that other resources may use alternate names (for example, the Negation Laws are sometimes known as the Law of Contradiction and the Law of the Excluded Middle, respectively).

Table 7: Some Logical Equivalences using AND ($\wedge$) and OR ($\vee$)

(a)	$p \wedge p \equiv p$	Idempotent Laws
	$p \vee p \equiv p$	
(b)	$p \wedge \mathbf{F} \equiv \mathbf{F}$	Domination Laws
	$p \vee \mathbf{T} \equiv \mathbf{T}$	
(c)	$p \wedge \mathbf{T} \equiv p$	Identity Laws
	$p \vee \mathbf{F} \equiv p$	
(d)	$p \wedge q \equiv q \wedge p$	Commutative Laws
	$p \vee q \equiv q \vee p$	
(e)	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associative Laws
	$(p \vee q) \vee r \equiv p \vee (q \vee r)$	
(f)	$p \wedge (q \vee r) \equiv (p \wedge q) \vee (p \wedge r)$	Distributive Laws
	$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$	
(g)	$p \wedge (p \vee q) \equiv p$	Absorption Laws
	$p \vee (p \wedge q) \equiv p$	

Table 8: Some More Logical Equivalences (adding ¬)

(a)	$\neg(\neg p) \equiv p$	Double Negation
(b)	$p \wedge \neg p \equiv \mathbf{F}$	Negation Laws
	$p \vee \neg p \equiv \mathbf{T}$	
(c)	$\neg(p \wedge q) \equiv \neg p \vee \neg q$	De Morgan's Laws
	$\neg(p \vee q) \equiv \neg p \wedge \neg q$	

Table 9: Some More Logical Equivalences (adding →)

(a)	$p \to q \equiv \neg p \vee q$	Law of Implication
(b)	$p \to q \equiv \neg q \to \neg p$	Law of the Contrapositive
(c)	$p \equiv \mathbf{T} \to p$	"Law of the True Antecedent"
(d)	$\neg p \equiv p \to \mathbf{F}$	"Law of the False Consequent"
(e)	$p \to p \equiv \mathbf{T}$	Self-implication (a.k.a. Reflexivity)
(f)	$p \to q \equiv (p \wedge \neg q) \to \mathbf{F}$	Reductio Ad Absurdum
(g)	$\neg p \to q \equiv p \vee q$	
(h)	$\neg(p \to q) \equiv p \wedge \neg q$	
(i)	$\neg(p \to \neg q) \equiv p \wedge q$	
(j)	$(p \to q) \vee (q \to p) \equiv \mathbf{T}$	Totality
(k)	$(p \wedge q) \to r \equiv p \to (q \to r)$	Exportation Law (a.k.a. Currying)
(l)	$(p \wedge q) \to r \equiv (p \to r) \vee (q \to r)$	
(m)	$(p \vee q) \to r \equiv (p \to r) \wedge (q \to r)$	
(n)	$p \to (q \wedge r) \equiv (p \to q) \wedge (p \to r)$	
(o)	$p \to (q \vee r) \equiv (p \to q) \vee (p \to r)$	
(p)	$p \to (q \to r) \equiv q \to (p \to r)$	Commutativity of Antecedents

Table 10: Some More Logical Equivalences (adding $\oplus$ and $\leftrightarrow$)

(a)	$p \leftrightarrow q \equiv (p \to q) \wedge (q \to p)$	Definition of Biimplication
(b)	$p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$	
(c)	$p \leftrightarrow q \equiv \neg p \leftrightarrow \neg q$	
(d)	$p \oplus q \equiv (p \wedge \neg q) \vee (\neg p \wedge q)$	Definition of Exclusive Or
(e)	$p \oplus q \equiv \neg(p \leftrightarrow q)$	
(f)	$p \oplus q \equiv p \leftrightarrow \neg q \equiv \neg p \leftrightarrow q$	

1.5.3 Applications of Logical Equivalences

Having a few dozen logical equivalences is nice, but what can we do with them? In a later chapter we will see how they help with arguments, but for now we can use them to verify additional equivalences, tautologies, and contradictions.

Example 27:

Problem: Using a truth table, demonstrate that $p \rightarrow (q \vee p)$ is a tautology.

Solution: We are doing this as a reminder: You don't need to use logical equivalences to demonstrate these relationships.

p	q	$[p]$	$q \vee p$	$p \rightarrow (q \vee p)$
T	T	[T]	T	**T**
T	F	[T]	T	**T**
F	T	[F]	T	**T**
F	F	[F]	F	**T**

As $p \rightarrow (q \vee p)$ evaluates to true for all possible combinations of values of p and q, it is a tautology.

We copied over the p column so that the q and p columns are in the same order as those operands are in $q \vee p$. This isn't a necessary step, especially not for operators like $\vee$ that are commutative, but we think that it is a good habit to acquire. Many people have a hard time accurately re-ordering operands of implication (which is not commutative) in their heads.

Note that you can say that the truth table shows that $p \rightarrow (q \vee p)$ is equivalent to true (that is, $p \rightarrow (q \vee p) \equiv \mathbf{T}$). That is a symbolic way of saying that the expression is a tautology.

Let us repeat that example, but this time we will apply logical equivalences.

Example 28:

Problem: Using a sequence of logical equivalences, demonstrate that $p \rightarrow (q \vee p)$ is a tautology.

Solution: The statement of the problem tells us where we need to go: We need to show that the proposition is equivalent to true. Can we find a known equivalence that both matches the form of $p \rightarrow (q \vee p)$ and seems a likely route to our desired answer? Yes! Table 9, line n, has the same form. True, it uses p, q, and r, but as these are just place-holders for propositions, we can substitute p for r. After applying that equivalence, the next two steps are straight-forward.

$$\begin{aligned} p \rightarrow (q \vee p) &\equiv (p \rightarrow q) \vee (p \rightarrow p) && \text{[Table 9(n)]} \\ &\equiv (p \rightarrow q) \vee \mathbf{T} && \text{[Self-Implication]} \\ &\equiv \mathbf{T} && \text{[Domination]} \end{aligned}$$

Because $p \rightarrow (q \vee p)$ is equivalent to true, it is a tautology.

Justifying each equivalence in the sequence is essential if the reader is to trust that the demonstration is correct. Writing one equivalence per line gives space to the right to either direct the reader to one of the tables on pages 33–34, or by giving its name (if it has one).

Frequently, there are other ways to string together equivalences to reach the same conclusion. Example 29 repeats Example 28 but chooses a different starting equivalence.

Example 29:

Problem: Using a sequence of logical equivalences different from that used in Example 28, demonstrate that $p \rightarrow (q \vee p)$ is a tautology.

Solution: There are no other given logical equivalences that have the same form. We have to be more creative. It might be tempting to just use self-implication directly, but its form $(p \rightarrow p)$ shows that the same proposition must appear on both sides of the implication. Instead, we will start with the Law of Implication (Table 9(a)) and see where that leads us.

$$\begin{aligned} p \rightarrow (q \vee p) &\equiv \neg p \vee (q \vee p) && \text{[Law of Implication]} \\ &\equiv q \vee (\neg p \vee p) && \text{[Commutativity of } \vee\text{]} \\ &\equiv q \vee \mathbf{T} && \text{[Negation Law]} \\ &\equiv \mathbf{T} && \text{[Domination]} \end{aligned}$$

Once again, as $p \rightarrow (q \vee p)$ is equivalent to true, it is a tautology.

The second step is actually more than just one step. We remove the parentheses, which is acceptable because $\vee$ is associative. Next, we swap (commute) $\neg p$ and q, and we insert parentheses (associativity of $\vee$ again) just to highlight the location of our next step. If you do not care for this lack of complete clarity, you can include a line for each of these steps. However, this kind of short-cutting is very commonly done, and so you should be prepared to mentally fill-in these small steps when studying examples.

At the risk of over-doing this problem, we wish to show one more technique for solving it. This approach is still based on logical equivalences, but considers the possible values of the operands in turn to reason about the solution. We call the technique *case-based reasoning*.

case-based reasoning

Example 30:

Problem: Demonstrate that $p \rightarrow (q \vee p)$ is a tautology by using case-based reasoning.

Solution: Being a logical proposition, p can be either true or false. We consider each in turn.

Case 1: Let p be true. Our expression becomes $\mathbf{T} \rightarrow (q \vee \mathbf{T})$. By domination, this simplifies to $\mathbf{T} \rightarrow \mathbf{T}$, which is true by the definition of implication.

Case 2: Let p be false. $\mathbf{F} \rightarrow (q \vee \mathbf{F})$ reduces to $\mathbf{F} \rightarrow q$ by the identity laws (Table 7(c)). By the definition of implication, any implication with false as the antecedent evaluates to true.

Thus, for both possible values of p, the proposition $p \rightarrow (q \vee p)$ is true, demonstrating that it is a tautology.

We will see this type of case-based reasoning again when we study proofs.

Knowledge of logical equivalences can help you make your programs more readable by other programmers (yes, this is a good thing!).

Example 31:

Problem: Write a condition that evaluates to true when the variable `sensor` is outside of the acceptable tolerance range defined by the constants `ZERO` and `OMG`. Assume that `OMG` > `ZERO`.

Solution: Assuming that a reading of `OMG` is unacceptable, 'outside' means that the value is either lower than `ZERO` or at/above `OMG`. That is, the sensor reading is not within the acceptable range. We can code that as:

```
if sensor < ZERO or sensor => OMG then ...
```

This is perfectly correct, but the code would be better if the condition could be more easily 'read' to mean "not acceptable" by the programmers who will be tasked with maintaining your code (and remember, one of them could well be you!). In logic, we can express that condition as $p \vee q$, with p : `sensor` $<$ `ZERO` and q : `sensor` $\geq$ `OMG`. $p \vee q \equiv \overline{\overline{p \vee q}}$ by double negation (Table 8(a)). Applying the second De Morgan's Law (Table 8(c)) to the inner negation produces $\overline{\overline{p} \wedge \overline{q}}$. The negation of p is `sensor` $\geq$ `ZERO` and the negation of q is `sensor` $<$ `OMG`. As a program condition, it looks like ...

```
if not (sensor >= ZERO and sensor < OMG) then ...
```

... and is logically equivalent to the first version. Yes, this has one more operator (the 'not') than does the first version, but it is a direct translation of "not acceptable" into code. Code that 'reads' well ("if the measurement is not acceptable, do something!") is more likely to be understood, and more likely to be maintained correctly. Give this some thought the next time you choose a cryptic identifier name (yes, like 'OMG') or receive a CT scan.[20]

One more example. We have already seen that conversions between English and logic can be challenging. Example 32 demonstrates how your knowledge of logical equivalences can help make sure your conversion to logic is correct.

Example 32:

Problem: Express this sentence in logical notation: *If he is honest, he didn't open his eyes and sneeze.*

Solution: Thinking positively, we start by identifying and labeling our

[20] One of the most famous examples of a software failure of a medical device is the Therac 25 story: http://en.wikipedia.org/wiki/Therac-25

predicates: h : *he is honest*, o : *he did open his eyes*, and s : *he did sneeze*. On the surface, the sentence is ambiguous. We don't know if *he didn't open his eyes and sneeze* should be expressed as $\overline{o} \wedge \overline{s}$ (*he did not open his eyes and he did not sneeze*, meaning that neither action occurred) or as $\overline{o \wedge s}$ (*he did not both open his eyes and sneeze*, meaning that he could have done either one separately or neither). Somehow, we have to choose between $h \rightarrow (\overline{o} \wedge \overline{s})$ and $h \rightarrow \overline{o \wedge s}$ as our answer. Perhaps logical equivalences can help us decide between them.

We know that the contrapositive of an implication is equivalent to the original (see Table 9). Considering our first option, we find that $h \rightarrow (\overline{o} \wedge \overline{s}) \equiv \overline{\overline{o} \wedge \overline{s}} \rightarrow \overline{h} \equiv \overline{\overline{o \vee s}} \rightarrow \overline{h} \equiv (o \vee s) \rightarrow \overline{h}$ (by contraposition, De Morgan's, and double negation). In conversational English, this could be written as: *If he opened his eyes or sneezed, he's lying.* This version does not make much sense; disjunction provides two or three options for truth, making lying unlikely.

Moving to the second option, applying contraposition and double negation to it shows that $h \rightarrow \overline{o \wedge s} \equiv (o \wedge s) \rightarrow \overline{h}$. In English: *If he opened his eyes and sneezed, then he is lying.* That is a reasonable statement; you could sneeze with your eyes open, but you'd almost certainly have to physically hold them open to do so.[21]

Thanks to logical equivalences, the correct logical interpretation appears to be $h \rightarrow \overline{o \wedge s}$.

Remember: *Do not overthink conversions between natural languages and logic!* It is easy to ask, "What if he opens his eyes, pauses, and closes them to sneeze?" Easy, but unjustified. Take the statement as given; don't waste time dreaming up embellishments. This material is challenging enough as-is.

[21] Thank you, Mythbusters! http://dsc.discovery.com/fansites/mythbusters/db/human-body/sneezing-eyeballs-pop-out.html