The first topic of this book deals with logical symbols and their manipulations. We will also introduce various other kinds of logical operators as well. The study of such manipulations has great connections with the notion of **mathematical proof**, which is an essential theme of this book (from Chapter 4 on, most of the concepts and problems will be of things that are *proven*, not computed). These operators will be useful in later parts of the book. To introduce these concepts we will motivate them by first discussing some physical analogues of these concepts that appear frequently in daily life.

1 Truth and Falsity

Consider what it means for a statement to be true (and respectively, to be false). One common idea is that statements that are true represent a fact. For example, it is true that upon first writing this book, the author was still living with his parents. The statement that the author is filthy rich and living the high life in Las Vegas is a false statement. In general, to assert the validity or invalidity of a statement, one must consider the objects in question and verify that whatever is asserted in the statement holds for them. For example, in order to confirm that the author is not filthy rich and living the high life in Las Vegas, one would check that the author does not make much money by checking their tax statements and check that he does not indeed live or frequent Las Vegas to have fun in morally dubious ways.

For the rest of this book we will not be concerned with any sort of epistemological justification for why certain statements are true or false. We will simply work with objects that are simple enough that we will be satisfied that certain statements can be taken by default to be true or false if necessary, or we will be able to determine the truth or falsity of these claims. We will also simplify our notion of truth to be *absolute*: to simplify our assumptions we will assume that any statement is either true, false, or ambiguous. In general we need to distinguish ambiguous statements. For example, consider the following statement:

$$2 \oplus 2 = 4.$$

Colloquially, one would read this as "2 oplus 2 equals 4". But the truth of this statement is ambiguous, for one, because it depends on how the symbol $\oplus$ is defined. Suppose I defined it in the following way:

$$a \oplus b = \begin{cases} a+b & b < a \\ a-b & b \geq a \end{cases}.$$

Then if we are working with integers or any number system containing the integers this statement is false. If instead we defined it a little differently as

$$a \oplus b = \begin{cases} a+b & b \leq a \\ a-b & b > a \end{cases}$$

then the previous statement would be true.

Now that we have motivated the notion of truth, in this section we will develop means of abstractly manipulating truth and falsity. The most abstract notion of this is denoted by the term **boolean algebra**. To make this notion more precise, we will introduce some terminology. There are only two values that we will work with explictly. These are true, denoted by T, and false, denoted by F. Some people prefer to work instead with the value 1 and 0, respectively. Whichever we choose, these two values are called boolean values. When refering to an expression that is one of T or F, we will call such an expression a boolean expression. We will indicate examples of these later in this section.

In general we will be concerned with manipulation of expressions where one can change an F in the expression to T, and vice versa. This is because we are interested in when certain patterns of boolean expressions are always equivalent to one another (that is, always both T or both F). For this purpose we will develop the notion of a boolean variable.

Definition 1. A boolean variable p represents a boolean value T or F. Usually this is ambiguous and not explicit. We denote an explicit assignment of a boolean value to a boolean variable using the $\equiv$ symbol. For example,

$$p \equiv T$$

denotes the explicit assignment from the boolean variable p to the boolean value T. Usually we will use the letters p, q, r, and so on to represent boolean variables.

If p is a boolean variable assigned to a boolean value, we define $\sim p$ by the other boolean value. So if $p \equiv T$, then $\sim p \equiv F$, and vice versa.

The full study of boolean algebra is the manipulation of boolean variables and boolean values using what are known as boolean operators. We will give an example of such an example below before introducing the definition.

Example 1. Given two boolean variables p and q, the expression $p \wedge q$ is defined as follows:

$$p \wedge q = \begin{cases} T & p \equiv T, q \equiv T \\ F & \text{otherwise.} \end{cases}$$

The symbol $\wedge$ is known as the **logical and operator**. It is a binary operator (that is, it acts on two variables p and q).

Why would we introduce such an operator in the first place? This is related to the colloquial use of "and" in everyday language. Intuitively, a statement of the form "P and Q" is only true if P and Q are both true. The $\wedge$ operator simply reflects this statement.

Definition 2. A binary logical operator $\oplus$ is an operation that takes two boolean values p and q and outputs depending on the values p and q a boolean value denoted $p \oplus q$.

A unary operator Δ is the same as a binary logical operator, but instead of two boolean values it takes one boolean value p and outputs depending on the value p a boolean value Δp.

As we can verify above, the logical and operator is an example of a binary logical operator. The negation operation $\sim$ is an example of a unary logical operator. We will give more examples of logical operators we will study in depth below.

p	q	$p \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

Given two boolean variables p and q, there are only finitely many combinations of boolean values that p and q can evaluate to. This means that we can tabulate the values of $p \oplus q$ in a finite table which completely describes the logical operator $\oplus$. Such a table is called a **truth table**. Below is the truth table for $\wedge$.

Here is how to interpret this table: each row's last entry denotes the value of $p \wedge q$ given the values for columns p and q. We tabulate the value of $p \wedge q$ given every combination of T and F that can be assigned to p and q together.

In general, truth tables can be extended to tabulate expressions of different variables. For example, here is a truth table for the boolean expression $(p \wedge q) \wedge r$.

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

For this table, observe that for a complete table we needed to record all possible values for p, q, and r, of which there are 8. In general if you have a logical expression with n variables the truth table for this expression will have 2^n rows.

In the following section we will start to indicate binary operators of interest (so that interesting logical expressions can be formed and studied).

2 A List of Interesting Binary Logical Operators

We have already seen the logical and binary operator $\wedge$ and the unary logical negation operator $\sim$.

2.1 The logical or ($\vee$) operator

The next operator we will introduce is the logical or operator $\vee$. The motivation for considering this operator is as follows. In colloquial language, a statement of the form "A or B" is usually taken to mean "either A or B is true." However, this excludes the scenario where A and B are both true. Taking this to account, the truth table for $\vee$ is the following:

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

2.2 The NAND and NOR operators

The NAND and NOR operators, $\uparrow$ and $\downarrow$ respectively can be defined in terms of previously defined operators. The NAND operator $\uparrow$ is defined as

$$p \uparrow q \equiv {\sim}(p \wedge q)$$

and the NOR operator $\downarrow$ is defined as

$$p \downarrow q \equiv {\sim}(p \vee q).$$

In other words, NAND and NOR are simply the logical negations of the results of the logical and and logical or, respectively. The reader should create truth tables for these operators if they want more practice in creating truth tables.

2.3 The XOR and XNOR operator

As we have stated before, the or operator does not reflect colloquial language of the term, which more precisely reflects the term "either-or". This operator is true whenever exactly one of its arguments is true, and false otherwise. It is denoted by $\oplus$. The truth table for this operator is written below:

p	q	$p \vee q$
T	T	F
T	F	T
F	T	T
F	F	F

The XNOR operator $\odot$ is defined to be the negation of the XOR operator. Later (In chapter 4) we will see that this operator plays an important role in defining what are called biconditional statements.

2.4 The implies ($\implies$) operator

The truth table of the implies operator $\implies$ is defined below.

The motivation of this operator is to tabulate situations that can happen given that any given implication statement is true. Suppose that the following statement is true no matter what

If it is raining outside, then Bob will carry an umbrella.

p	q	$p \implies q$
T	T	T
T	F	T
F	T	F
F	F	T

Then consider the statements "it is raining outside" and "Bob is carrying an umbrella". Consider the following scenarios.

- It is possible for it to be both raining outside and for Bob to be carrying an umbrella at the same time, for the statement above is always true. So both statements being true is possible.
- It is possible for it to be not raining outside and yet for Bob to be carrying an umbrella. For what if Bob carried an umbrella all the time? The statement above does not discount that possibility. So for the first statement to be true and the second to be false is possible.
- It is not possible for it to be raining outside and Bob to not be carrying an umbrella, as the statement given forbids this possibility.
- It is possible for it to be not raining outside and for Bob to be not carrying an umbrella. For the statement does not say anything about Bob when it is not raining outside.

For every case where the two statements are possible we let the implication operator record true, and for the one case where the two statements are not possible we let the implication operator record false.

3 Evaluation of Boolean Expressions as Functions

In this section we highlight the notion of a boolean expression as a function. Using this perspective it will be clear what it means for two expressions to be logically equivalent.

A function will take a value (a number, an animal, anything) and output another value (a letter of the alphabet, the number of the animal's legs, anything). Later on we will learn a more rigorous definition than that described above.

Then a boolean expression is really a function. Each variable takes in either true or false, at which point we can *evaluate* the expression and we get some boolean value (either true or false). Consider the expression

$$(p \wedge q) \vee r.$$

Then we can feed in any combination of trues or falses. For example, if $p \equiv T$, $q \equiv F$, and $r \equiv T$, then the expression above will evaluate these values to

$$(T \wedge F) \vee T \equiv F \vee T \equiv T,$$

or true.

A **truth table** is then really just a way of recording the output values of this function. We summarize our findings here:

Two logical expressions are called **logically equivalent** if they have the same truth value given a set of truth values for each variable.
Equivalently, two logical expressions are logically equivalent if they have the same truth table.

Let's consider the special case where our logical expression is $p \otimes q$ for a binary logical operator $\otimes$. This can be seen as a function of p and q. So we might write it using notation like

$$O(p,q) = p \otimes q.$$

For instance, we might consider the expression $(p \wedge q) \vee r$ once again. Then we can view this expression in the notation

$$\mathrm{OR}(\mathrm{AND}(p,q),r).$$

This notation is also useful as it makes unambiguous how expressions are evaluated. This brings us to the topic of when expressions are ambiguously defined. For example, consider the expression $p \wedge q \vee r$. This is ambiguously defined. It could mean the following two expressions (when given in functional notation):

$$\mathrm{OR}(\mathrm{AND}(p,q),r) \tag{1}$$
$$\mathrm{AND}(p,\mathrm{OR}(q,r)). \tag{2}$$

These expressions are not the same! So the expression being considered is called "ambiguous". This makes clear that when evaluating a logical expression it matters which part of the expression you evaluate first. However, there are some expressions where this does not matter. For example, the logical expression $p \wedge q \wedge r$ is unambiguous, because no matter if you take this expression to be $(p \wedge q) \wedge r$ or $p \wedge (q \wedge r)$, the resulting truth table remains the same. This pattern with repeated ands turns out to be unambiguous for any number of variables, but showing that this is true is somewhat difficult (and will be addressed as a topic in a later chapter).

In general, logical operators follow a certain pattern which lets us recursively evaluate them. Such evaluation lets modern computers (such as the one that this book was typeset) evaluate these expressions much like a computer would. For simplification we will assume that our logical expression consists of only boolean variables (like p, q, etc.), $\wedge$, $\vee$, and $\sim$ (and as we will see in a following section this actually will not lose any generality). Then any logical expression is of one of the following forms:

- Some boolean variable p.
- $\mathrm{AND}(e_1, e_2)$, where e_1 and e_2 are logical expressions.
- $\mathrm{OR}(e_1, e_2)$, where e_1 and e_2 are logical expressions.
- $\mathrm{NOT}(e)$, where e is an expression.

Then we can proceed to evaluate the expressions, which are either one or two simpler logical expressions, or just a boolean variable which we evaluate to T or F depending on what evaluation we wanted to do.

In general computers, when interpreting a logical expression like

$$(p \wedge q) \vee r$$

will convert this expression into function notation as above, and then evaluate the expression once it has been converted using the rules above.

1. Create truth tables for the following expressions.

   - $p \wedge (\sim p)$ (This formula is called *unsatisfiable*. Why is this?)
   - $p \wedge T$ (Here, T represents true. Similarly, F represents false).
   - $p \wedge F$
   - $p \vee T$ and $p \vee F$.

2. Let's consider the logical expressions in problem 1 again. Figure out simpler expressions which are equivalent to these.

3. Consider the following chunk of code. Here, `var1`, `var2` are boolean variables (ie, they are true or false).

   ```
   if (var1 && (var1 || var2)) { printf("She loves me!"); }
   else { printf("She loves me not..."); }
   ```

   Simplify the code.

4. We have made it a point in this section that given a logical expression (or even an arithmetic expression), parentheses matter!

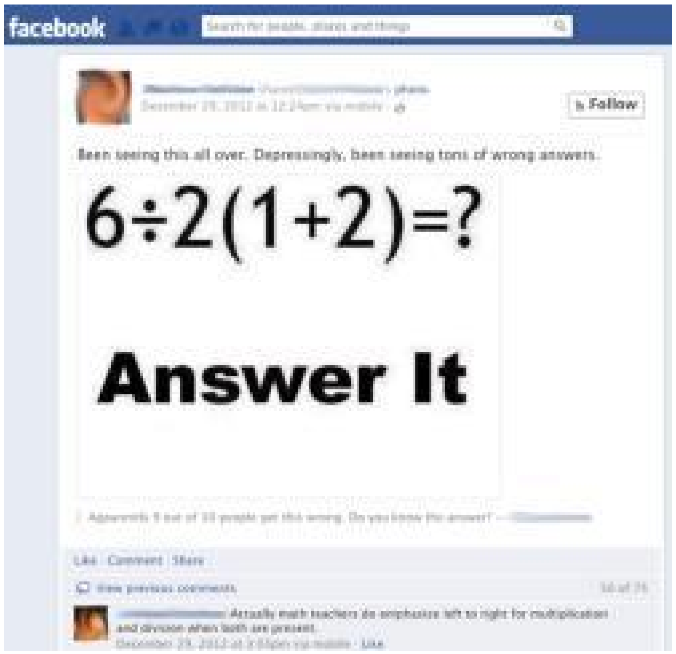
   Build the truth tables for $(\sim a) \wedge b$ and $\sim (a \wedge b)$. Compare it to the viral facebook math problem below:

5. This problem mostly deals with the implies ($\implies$) logical operator. The motivation behind this operator is that eventually we want to do **proofs**. That is, given a set of initial hypotheses, we want to make deductions and logically deduce that a conclusion is true. For example, eventually all of you will be able to prove, roughly stated:

   $$(n \text{ is even}) \implies (n+1 \text{ is odd}).$$

   Construct the truth tables for all the logical expressions below.

   - $p \implies (p \wedge q)$ and $p \implies (p \vee q)$.
   - $(p \wedge q) \implies (p \vee q)$. This formula is an example of a *tautology*.
   - $(p \vee q) \implies (p \wedge q)$.



Been seeing this all over. Depressingly, been seeing tons of wrong answers.

6÷2(1+2)=?

Answer It

Apparently 9 out of 10 people get this wrong. Do you know the answer?

[redacted] Actually math teachers do emphasize left to right for multiplication and division when both are present.

Figure 1: Viral Facebook Math Meme

- $(p \implies q) \implies (p \iff q)$.
- $(p \iff q) \implies (p \implies q)$.
- $p \implies q \implies r$ and $p \implies r$.

6. Convert the following expressions into functional notation. If the expression is ambiguous, then list all possible functions which the expression could represent. Use NAND for the $\uparrow$ operator and NOT for the $\sim$ operator. (For a unary operator, NOT takes how many variables?)

   - $(p \uparrow q) \vee (q \wedge p)$
   - $\sim p \vee q$
   - $p \vee q \uparrow r \wedge s$

7. In this problem we will outline how one can write a computer program that can parse and evaluate logical expressions such as $\sim(T \vee F) \wedge (F \vee T)$. For the sake of simplicity, we may assume that any logical expression only consists of the standard and ($\wedge$), or ($\vee$), and not ($\sim$) operators.

   (a) Scan the input string and form a "token list" of all the individual relevant characters. For example, an expression of the form $\sim(T \wedge F) \vee F$ can be decomposed into the form

   ```
   [~, (, T, ^, F, ), v, F]
   ```

(b) Assuming that your expression is well founded, one can make what is known as a *context free grammar*, which is a set of rules that recursively define your expression.

(c) Write a set of computer functions that call each other and parse your token list, based on your recursive expression definition. In the language you are programming in you will need to make structures that are of the form $f(a, b)$, in functional notation. For example, in python, one could potentially use heterogeneous arrays, for example,

```
[‘‘and’’, [‘‘or’’, true, false], false]
```

to represent these expressions. The resulting expression that comes out is known as a “syntax tree”, and closely models the functional expressions described in the text.

(d) Write a function that will recursively evaluate a syntax tree that has been written, as described in the chapter.

4 Various Relationships between Logical Operators

In this section, we will indicate several important logical equivalences between the logical operators that we have introduced thus far.

Recall in the previous section that boolean expressions can be viewed as functions of each variable, so we deduced that two expressions are logically equivalent if they have the same truth table. In fact, in some cases we can compare boolean expressions with different number of variables. For example, the logical expression $(p \wedge q) \vee (r \vee (\sim r))$ is logically equivalent to the logical expression $p \wedge q$. We can naturally view $p \wedge q$ as a function of the boolean variables p and q, but we can also view this as a function of p, q, and r, where the result is independent of the value of r. So in this way we can compare logical expressions with varying or different boolean variables.

Now we will show a basic logical equivalence called the **contrapositive law**. The contrapositive law asserts for any logical variables p and q we have the equivalence

$$p \implies q \equiv (\sim p) \implies (\sim q).$$

To demonstrate that this equivalence is true it suffices to show that each entry in their respective truth tables are equal, as demonstrated below:

p	q	$p \implies q$		p	q	$(\sim q) \implies (\sim p)$
T	T	T		T	T	T
T	F	T		T	F	T
T	F	F		T	F	F
F	T	T		F	T	T

The same approach is used when demonstrating each logical equivalence in the table below.

Logical Equivalence	Description
$p \wedge q \equiv q \wedge p$, $p \vee q \equiv q \vee p$	Commutativity of $\wedge$ and $\vee$
$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$	Associativity of $\wedge$
$(p \vee q) \vee r \equiv p \vee (q \vee r)$	Associativity of $\vee$
$(p \vee q) \wedge r \equiv (p \wedge r) \vee (q \wedge r)$	Distributivity of $\wedge$ over $\vee$
$(p \wedge q) \vee r \equiv (p \vee r) \wedge (q \vee r)$	Distributivity of $\vee$ over $\wedge$
$p \wedge T \equiv p$	$\wedge$ identity
$p \vee F \equiv p$	$\vee$ identity
$\sim(\sim p) \equiv p$	Double Negation
$p \wedge p \equiv p \vee p \equiv p$	Idempotence of $\wedge$ and $\vee$
$p \vee (p \wedge q) \equiv p \wedge (p \vee q) \equiv p$	Absorption Property
$a \implies b \equiv (\sim b) \implies (\sim a)$	Contrapositive Property
$a \implies b \equiv (\sim a) \vee b$	Implication as Disjunction
$a \odot b \equiv (a \implies b) \wedge (b \implies a)$	XNOR as Conjunction
$a \wedge (\sim a) \equiv F$, $b \vee (\sim b) \equiv T$	Simplification Rules

Using logical equivalences like the ones in the table above, it is now possible to show that certain expressions are logically equivalent without explicitly constructing their truth tables, as the examples below show.

Example 2. In this example we will show that the expression

$$((q \vee z) \wedge (k \vee (\sim k))) \implies (((\sim q) \wedge (\sim z)) \vee p)$$

is logically equivalent to the simpler expression

$$(\sim(q \vee z)) \vee p.$$

To do this we will simplify the first expression using rules stated in the table.

$$\begin{aligned}
&((q \vee z) \wedge (k \vee (\sim k))) \implies (((\sim q) \wedge (\sim z)) \vee p)\\
\equiv\, &((q \vee z) \wedge T) \implies (((\sim q) \wedge (\sim z)) \vee p)\\
\equiv &(q \vee z) \implies (((\sim q) \wedge (\sim z)) \vee p)\\
\equiv &(q \vee z) \implies (\sim(q \vee z) \vee p)\\
\equiv &(\sim(q \vee z) \vee \sim(q \vee z)) \vee p\\
\equiv &(\sim(q \vee z)) \vee p
\end{aligned}$$

as desired.

Example 3. In this example we will simplify the expression

$$(k \wedge l) \vee ((k \wedge m) \wedge k \wedge (l \vee (\sim m))).$$

But not now because I am lazy.

In general, the rules above (and rules that can be derived from them) form the basis of a system where one can do calculations with logical variables. Now we will discuss some of these properties individually and make more comments about them.

4.1 Commutativity and Associativity

In general, a binary operator $\otimes$ is said to be commutative if $p \otimes q \equiv q \otimes p$, and associative if $(p \otimes q) \otimes r \equiv p \otimes (q \otimes r)$. As noted in the table above, the operators $\vee$ and $\wedge$ are associative and commutative.

Commutativity and Associativity are rather strong properties to have for any binary operator. This is because not all operators are associative or commutative. For example, the implies operator is not commutative, that is,

$$p \implies q \not\equiv q \implies p,$$

and the NAND operator is not associative, that is

$$p \uparrow (q \uparrow r) \not\equiv p \uparrow (q \uparrow r).$$

One can see this by constructing the truth table for each expression respectively.

4.2 The Absorption Property

The proper way to interpret the absorption properties in the table above are that their values are independent of the variable q when evaluated. To see this, consider the expression $p \vee (p \wedge q)$. Evaluating q as F we get

$$p \vee (p \wedge F) \equiv p \vee F \equiv p$$

by other properties in the table. Similarly, evaluating q as T we get

$$p \vee (p \wedge T) \equiv p \vee p \equiv p.$$

4.3 Implication as Disjunction

The implication as disjunction property is the first example of the important idea of creating an expression given a truth table where all the variable assignments evaluate to true except for one. In this case, the implies operator evaluates T unless $p \equiv T$ and $q \equiv F$. In this case, we can create an or expression $(\sim p) \vee q$ which we see has the same truth table as $p \implies q$. This idea of creating such an indicator function given any truth table later will be important when we study the conjunctive and disjunctive normal forms in a following section.

4.4 Exercises

1. Consider the following expressions

$$p \implies (q \vee r), (p \wedge (\sim q)) \implies r, (p \wedge (\sim r)) \implies q.$$

Show that the above expressions are logically equivalent...

- directly by using a truth table.
- simplifying expressions using the rules in the table.

2. Simplify the expression

$$z \wedge ((q \vee ((\sim w) \wedge q)) \wedge ((\sim z) \vee (\sim q)))$$

as much as possible.

5 Disjunctive and Conjunctive Normal Forms

We have already seen that given any logical expression L with n variables we can form a truth table with precisely 2^n rows documenting the output of L when it is evaluated at each possible input. Now we want to answer the reverse question: if we have a table Y of the format of a truth table with 2^n rows documenting some possible output with each input of n variables, then is there a logical expression involving all n variables and some of the binary logical operators introduced earlier for which its truth table is exactly the table Y? It turns out that this answer is yes, and in this section we will see why.

5.1 The $\wedge$ as an indicator function

Suppose that $x_1, \ldots, x_n$ are our boolean variables under consideration. Then the logical expression

$$L \equiv x_1 \wedge x_2 \wedge \cdots \wedge x_n$$

evaluates to T only if every variable x_k is set to T. So L can be seen as an indicator function which is true on only one possible input of logical variables. But what if I wanted an expression which evaluates to T only if every variable except x_1 is set to T? This is not too much of a problem either, since the expression

$$K \equiv (\sim x_1) \wedge x_2 \wedge \cdots \wedge x_n$$

evaluates to T in this case and F otherwise. With these two examples it is clear how to make an "indicator function" which evaluates to true given one assignment of random variables and false otherwise. Simply take the expression

$$\bigwedge_{x_k \to F} (\sim x_k) \wedge \bigwedge_{x_j \to T} x_j$$

where the big $\wedge$ represents a logical and over all the variables in question where we want the variables to be F or T. It is clear that this expression evaluates to T only one one choice of variable assignment.

5.2 The $\vee$ as a join function

So far we have demonstrated how to represent any truth table Y with exactly one row of the table evaluating to T as a logical expression. However, what about the rest? The remaining questions can be answered once we consider the act of conjoining equations using the $\vee$ operator.

First consider the logical expression L in the previous subsection. For any other logical expression M, consider the logical expression $L \vee M$. One thing we can say about $L \vee M$ is that it evaluates to true when every variable x_k is set to T. We don't know that much about its evaluation at any of the other variables though. If instead we considered the logical expression $L \vee K$, where K was defined in the previous section, then we see that $L \vee K$ evaluates to T at exactly two assignments, precisely those that made L and K evaluate to T.

Using these two concepts it is now clear how to represent any truth table Y using a logical expression. First, consider for which assignments of variables Y evaluates to T. For these assignments create indicator $\wedge$ expressions L_j which evaluate to true only when we have that specific variable assignment. Finally, take all such expressions L_j and conjoin them with $\vee$ expressions to create a final expression.

Example 4. Consider the following truth table:

p	q	??
T	T	T
T	F	F
F	T	T
F	F	F

We will create a logical expression whose truth table is the same as this truth table. Note that the this table only evaluates to true when $p \equiv T, q \equiv T$, and $p \equiv F, q \equiv T$. The associated indicator $\wedge$ functions for these assignments are $p \wedge q$ and $(\sim p) \wedge q$. Hence the final expression whose truth table is equal to the above truth table is

$$(p \wedge q) \vee ((\sim p) \wedge q).$$

One can see that the process described above will always generate a logical expression whose truth table is equivalent to the given truth table. To see this, we observe that the expression we generate is several indicator expressions linked up with the $\vee$ operator. If any one of these is true, then the whole expression is true. Conversely, if an assignment of variables goes to false in the original truth table then it does not match any of the indicator expressions, so the whole logical expression evaluates to false. The final expression that we obtain is said to be in **disjunctive normal form**.

5.3 Using $\vee$ as an indicator function instead

Note that given n boolean variables $x_1, \dots, x_n$, the expression

$$x_1 \vee \cdots \vee x_n$$

evaluates to T for all variable assignments except for the assignment where every boolean variable is assigned F. Using similar ideas to the first subsection, we can create indicator logical expressions where every assignment evaluates to true except for one. To create a general expression we can take such indicator expressions and join them together with $\wedge$ taking in mind that any logical expression L will satisfy

$$L \wedge F \equiv F.$$

The resulting expression obtained is said to be in **conjunctive normal form**.

6 Boolean Satisfiability

In this section, we will briefly discuss a property of boolean formulas called **satisfiability**. This was briefly discussed in an exercise in a previous section, when considering the boolean formula $p \wedge (\neg p)$.

A boolean formula is **satisfiable** if there exists an assignment of true or false to each of its variables which makes the formula evaluate to true. Such an assignment is called a **satisfying assignment**. Note that we only need *one* such assignment of variables. A boolean formula is **unsatisfiable** if it isn't satisfiable. In particular, every assignment of variables will evaluate to false.

For example, the formula $p \wedge q \wedge r$ is satisfiable (with satisfying assignment $p \equiv q \equiv r \equiv T$. And this is the ONLY satisfying assignment). On the other hand, the formula $(x \vee F) \wedge ({\sim}x)$ is not satisfiable, as no matter whether x is T or F the expression evaluates to F either way.

To determine boolean satisfiability of a formula is an interesting problem which has caught the interest of computer scientists, so we will make some more heuristics related to the problem. First note that if a formula is satisfiable you only need to provide a satisfying assignment. So in order to demonstrate a formula with n boolean variables is satisfiable I only need the time to evaluate a single formula with n variables. However, if I wanted to show that a formula is unsatisfiable instead, I would have to evaluate 2^n different satisfying assignments. This number gets big really fast. In particular, when $n = 24$, even if I could evaluate one boolean formula every second it would still take me nearly a year to demonstrate a formula to be unsatisfiable.

In particular, it appears that demonstrating that a formula is satisfiable or unsatisfiable is a big problem: it seems like the only strategy is to try every satisfying assignment and hope you get lucky. We have seen already with $\wedge$ indicator functions that there might be only one satisfying assignment. So determining at a glance whether or not a formula is satisfiable seems to be a hard problem. There are some cases where this might be easy to do however, which are detailed in the exercises.

1. Determine whether or not the following Boolean formulae are satisfiable:
   - $x_1 \wedge x_2 \wedge \cdots \wedge x_n$, and $x_1 \vee x_2 \vee \cdots \vee x_n$, where n is a natural number.
   - $(x_1 \wedge (\neg x_2)) \vee (x_2 \wedge (\neg x_3)) \vee \cdots \vee (x_{n-1} \wedge (\neg x_n))$.
   - $(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$.
   - $(x_1 \vee (\neg x_2) \vee x_3) \wedge (x_1 \vee (\neg x_2) \vee (\neg x_3)) \wedge ((\neg x_1) \vee x_2 \vee (\neg x_3))$.

2. Verify the claim in the text that if $n = 24$ a formula a second would take nearly a year to confirm unsatisfiable. Note that a year has 365 days.

3. Sometimes, despite the computational time it takes to find a satisfying assignment, some unfortunate circumstance may require that you do so. This problem will outline one or two heuristics in order to possibly simplify your job a little further.

   Consider the boolean formula

   $$(x_1 \wedge x_5 \wedge x_2) \vee (x_3 \wedge (\neg x_3) \wedge x_5) \vee (x_2 \wedge (\neg x_7) \wedge x_6 \wedge x_1 \wedge (\neg x_4)).$$

   (a) Notice that this formula is in *disjunctive normal form.* That is, it can be represented as an OR of ANDS. Explain how this might simplify our problem a little bit.

   (b) Simplify the formula by considering the variable x_3.

   (c) Discuss (with your neighbors) the theoretical or practical use of such simplifications.