Logic and Induction

Will Rosenbaum

October 7, 2024

In this essay, we introduce the basic vocabulary and mechanics of symbolic logic. We adopt an informal approach which focuses on mechanical aspects and applications. We introduce the principle of mathematical induction and apply the principle to analyze the output of iterative and recursive algorithmic procedures.

1 Propositional Logic

At its most fundamental level, logic is the study of truth. A (logical) ***proposition*** is simply a declarative statement which could be either true or false. We will denote propositions with capital letters: $P, Q, R, \ldots$. For example, P could stand for the statement "x is a prime number," or "Alice enjoys studying math." One can think of propositions as variables which can assume one of two values: T for "true" or F for "false."

1.1 Logical Connectives

Logical connectives allow us to string together multiple propositions to form more complicated statements. The basic connectives—along with their corresponding symbols—are indicated in Figure 1.

symbol	name
$\wedge$	and (conjunction)
$\vee$	or (disjunction)
$\neg$	not (negation)
$\implies$	implies
$\iff$	if and only if (equivalence)

Figure 1: The basic logical connectives.

Using these connectives, we can form compound sentences from simple propositions. The statement $A \implies B$ can be read either as "A implies B" or "if A then B."

Example 1. *Suppose P is the statement "x is a prime number," Q is the statement "x is an even number," and R is "$x = 2$." We interpret the statement $P \wedge Q$ as "x is a prime number and x is an even number." Going further, we can analyze the statement*

$$(P \wedge Q) \implies R.$$

Here we use parentheses in order to indicate that the "$\wedge$" of P and Q is computed before the "$\implies$." We can interpret this as encoding the statement "if x is prime and x is even, then $x = 2$."

1.2 Truth tables

The nature of a compound proposition (i.e., a proposition including at least one connective) is elucidated by evaluating its ***truth table***. A truth table is an explicit computation of the truth values of the proposition for all possible values (true or false) of its sub-statements. In Figure 1 we give a truth table for the basic connectives given above. We think of the truth table as *defining* these connectives. Using this table, one can in principle evaluate any compound proposition for any value of its simple sub-propositions.

Exercise 2. *Compute the truth table for the following propositions:*

P	Q	$P \wedge Q$	$P \vee Q$	$\neg P$	$P \implies Q$	$P \iff Q$
T	T	T	T	F	T	T
T	F	F	T	F	F	F
F	T	F	T	T	T	F
F	F	F	F	T	T	T

Figure 2: The truth table for the basic logical connectives.

1. $P \wedge (\neg Q)$
2. $Q \implies (\neg P)$
3. $P \implies (\neg Q)$
4. $\neg(P \implies Q)$
5. $(\neg Q) \implies (\neg P)$

1.3 Tautologies and Logical Equivalence

A ***tautology*** is a compound proposition which evaluates to T for any truth assignments (T or F) to its sub-propositions. Equivalently, we can think of a tautology is a proposition whose truth table is always T.

Exercise 3. *Verify that the following propositions are tautologies:*

1. $P \vee (\neg P)$
2. $P \iff P$
3. $(P \wedge (\neg P)) \implies Q$

We say that two propositions P and Q are ***logically equivalent*** if $P \iff Q$ is a tautology. Logical equivalence is important because if two propositions P and Q are logically equivalent, then in order to demonstrate the truth of one, it suffices to establish the truth of the other. Thus logical equivalence can be used to justify *proof techniques*.

Example 4. *De Morgan's laws describe how negation interacts with conjunctions and disjunctions:*

1. $\neg(P \wedge Q) \iff (\neg P) \vee (\neg Q)$,
2. $\neg(P \vee Q) \iff (\neg P) \wedge (\neg Q)$.

These laws are easily verified by computing the truth tables for the two statements. For example

P	Q	$P \wedge Q$	$P \vee Q$	$\neg(P \wedge Q)$	$(\neg P) \vee (\neg Q)$	$\neg(P \wedge Q) \iff (\neg P) \vee (\neg Q)$
T	T	T	T	F	F	T
T	F	F	T	T	T	T
F	T	F	T	T	T	T
F	F	F	F	F	F	T

Since the final column is always T, the first of De Morgan's laws is indeed a tautology, hence $\neg(P \wedge Q)$ and $(\neg P) \vee (\neg Q)$ are logically equivalent.

Exercise 5. *Prove that $P \iff Q$ is logically equivalent to $(P \implies Q) \wedge (Q \implies P)$.*

2 Predicate Logic

So far, our logical framework serves only to evaluate the truth of propositions. In order to connect this logic to the larger body of mathematics, we require mathematical variables and quantifiers. ***Mathematical variables*** are symbols that correspond to mathematical objects (e.g., numbers, sets, etc.) rather than logical propositions. We typically use lower-case letters for mathematical variables, $x, y, z, \ldots$. For example, x might refer to an integer or a rational or real number.

A ***logical predicate*** defined on a universe of mathematical objects is a symbol that represents a property that each object in the universe may or may not have. Thus, if x is a mathematical variable, and P a predicate, $P(x)$ is the proposition that evaluates to T when x satisfies the predicate and F when x does not satisfy the property.

Example 6. *Let* $\mathbf{N}$ *denote the set of natural numbers (i.e.,* $\mathbf{N} = \{0, 1, 2, \ldots\}$*), and let* P *the predicate indicating if a number is even. That is,*

$$P(x) = \begin{cases} T & \text{if } x \text{ is even} \\ F & \text{otherwise.} \end{cases}$$

While P *is a predicate, for each natural number* x*,* $P(x)$ *is a* proposition *indicating whether or not* x *is even.*

2.1 Quantifiers

In addition to mathematical variables, standard predicate logic uses two ***quantifiers***:

1. $\forall$ read "for all,"
2. $\exists$ read "there exists."

Mathematical definitions are formed by stringing together one or more quantifiers each with an associated (mathematical) variable, followed by a proposition involving the variables. The quantifier $\forall$ is known as the ***universal quatifier***, while $\exists$ is the ***existential quantifier***. $\forall x \cdots$ indicates that the proposition following the universal quantifier is true for every possible value of x. Similarly $\exists x \ldots$ indicates that there is some value of x making the proposition true. We read $\exists x \ldots$ as "there exists x such that $\ldots$"

Example 7. *Here are some familiar mathematical definitions written in logical notation. Throughout* m, n *and* p *will denote natural numbers. We denote the set of natural numbers* $\mathbf{N} = \{0, 1, 2, \ldots\}$.

1. *n is even:*

$$(\exists m \in \mathbf{N})[n = 2m].$$

We read this statement as "there exists m *in* $\mathbf{N}$ *such that* $n = 2m$*."*

2. *m is divisible by n (which we denote by $n|m$):*

$$(\exists p \in \mathbf{N})[m = pn].$$

This reads "there exists p *in* $\mathbf{N}$ *such that* $m = pn$*."*

3. *p is a prime number:*

$$(\forall n \in \mathbf{N})(\forall m \in \mathbf{N})[((n \neq 1) \wedge (m \neq 1)) \implies p \neq mn]$$

We read this statement as "for all $n \in \mathbf{N}$ and for all $m \in \mathbf{N}$, if n and m are not equal to 1, then $p \neq mn$." This formally encodes the statement that p is not divisible by any number other than 1 and itself.

2.2 Negation of Quantifiers

It is often the case that we would like to negate some statement involving quantifiers. To do so, we apply the following rules:

1. $\neg((\forall x)P(x)) \iff (\exists x)(\neg P(x))$
2. $\neg((\exists x)P(x)) \iff (\forall x)(\neg P(x))$.

To negate statement with a quantifier, reverse the quantifier (i.e., $\forall$ becomes $\exists$ and vice versa) and negate the rest of the statement. If a statement contains multiple quantifiers, each of the quantifiers gets reversed.

For example, we can negate a doubly quantified expression $\forall x \exists y P(x,y)$ as follows:

$$\begin{aligned}\neg(\forall x \exists y P(x,y)) &\iff \exists x \neg(\exists y P(x,y)) \\ &\iff \exists x \forall y \neg P(x,y).\end{aligned}$$

Exercise 8. *Suppose S is the set of people in a society. We call a person $p \in S$ a **dictator** if for every person $q \in S$, q obeys p. The society is a **dictatorship** if S contains a dictator.*

1. *Write the definition of dictatorship using quantifiers and logical notation, where $P(q,p)$ is the predicate ""q obeys p."*
2. *Determine the negation of the expression "S is a dictatorship" in logical notation.*
3. *How would you express the negation of this expression in plain English?*

3 Mathematical Induction

Mathematical induction—or simply "induction"—is a logical principle that allows us to reason about sequences of events by analyzing individual events. Induction has a pervasive role in the analysis of algorithms in computer science. The central task of algorithm design is to devise an automated procedure that breaks each possible instance of a problem into a sequence of "elementary" operations.

To establish that an algorithm does indeed perform a prescribed task, we must argue that for *every* instance of the task, the algorithm produces the correct output. This may seem an impossible endeavor since, in principle, there could be infinitely many instances or inputs. The principle of induction allows us to reduce the problem of reasoning about entire executions of algorithms to reasoning about individual steps that an algorithm takes. Designing an algorithm is the process of breaking a task down into individual steps. Induction gives us a tool to argue that the individual steps fit together to solve the original problem.

Postulate (Principle of Induction)**.** *Suppose $P(0), P(1), P(2), \ldots$ is a sequence of predicates indexed by the natural numbers. Suppose we establish that*

1. *$P(0)$ is true (the **base case**), and*

2. *for all i, if $P(i)$ is true, then $P(i+1)$ is also true (**inductive step**).*

Then for every $n \in \mathbf{N}$, $P(n)$ is true. We can write the principle of induction in logical notation as

$$(P(0) \wedge \forall i(P(i) \implies P(i+1))) \implies \forall n(P(n))$$

The principle of induction formalizes the following line of reasoning. Suppose we wish to establish that $P(0), P(1), P(2), \ldots$ are all true. If we argue the base case (that $P(0)$ is true) and the inductive step (that whenever $P(i)$ is true, then so is $P(i+1)$), then we can reason as follows:

1. $P(0)$ is true because this is the base case.

2. Since $P(0)$ is true, then so is $P(1)$ by the inductive step with $i = 0$.

3. Since $P(1)$ is true, then so is $P(2)$ by the inductive step with $i = 1$.

4. Since $P(2)$ is true, then so is $P(3)$ by the inductive step with $i = 2$.

5. $\ldots$

While this reasoning may be intuitive, the principle of induction asserts that our conclusion—that all of the $P(n)$ are true—is a logically sound conclusion. In what follows, we use the principle of induction in order to justify our claims about the behavior of a few procedures.

3.1 Iterative Example

Consider the following method:

```
1: procedure IterativeSum(n)          ▷ Sum the numbers from 1 to n
2:     total ← 0
3:     for i = 1, 2, ..., n do
4:         total ← total + i
5:     end for
6:     return total
7: end procedure
```

Note that on input n, this method returns the sum of the numbers $1 + 2 + \cdots + n$. For large values of n, this method is pretty inefficient. We would like to find a better method (i.e., simple formula) for computing the method's output without having to perform all n iterations of the loop. We will show that, in fact, such a simple formula exists.

Proposition 9. *For every positive integer n,* IterativeSum(n) *returns the value $\frac{1}{2}n(n+1)$.*

We can verify the proposition by hand for a few small values of n. However, we cannot hope to establish the proposition by exhaustively checking inputs and outputs, since the proposition's conclusion must hold for all (of the infinitely many!) positive integers. Thus, it is natural to try to argue by induction.

Towards an argument by induction, we must decide precisely what claim it is we are making about the method IterativeSum. When

analyzing an iterative method (i.e., a method containing a loop), it is often a good strategy to find a ***loop invariant***, i.e., some property that the loop maintains before and after each iteration. We can perform a few iterations of the loop by hand to see what is going on with total:

- before the first iteration total = 0
- after iteration $i = 1$, total = 1
- after iteration $i = 2$, total = $1 + 2 = 3$
- after iteration $i = 3$, total = $1 + 2 + 3 = 6$
- after iteration $i = 4$, total = $1 + 2 + 3 + 4 = 10$
- …

Observe that the pattern of values of total is $1, 3, 6, 10, \ldots$, which are precisely the values of $\frac{1}{2}n(n+1)$ for $n = 1, 2, 3, 4, \ldots$. Thus, we are lead to conjecture the following:

Claim 10 (Loop Invariant)**.** *For every positive integer i, after iteration i of the loop in* ITERATIVESUM, total *stores the value $\frac{1}{2}i(i+1)$.*

We will use induction to prove the loop invariant. Before writing up the argument, however, we need to do a bit of scratch work. The base case of the argument ($i = 1$) is straightforward, but we need to see how to derive the inductive step. The key is in the assignment total ← total + i. In iteration $i + 1$, total ← total + $i + 1$. Again, what we are required to show is that if the claim (loop invariant) holds after iteration i, then it also holds after iteration $i + 1$.

Supposing the claim holds after iteration i, we have total = $\frac{1}{2}i(i+1)$. Then in iteration $i + 1$ we update total ← total + $i + 1$. By the **inductive hypothesis** (that total = $\frac{1}{2}i(i+1)$ before this operation) we compute:

$$\begin{aligned}
\text{total} &= \frac{1}{2}i(i+1) + i + 1 \\
&= \frac{1}{2}i^2 + \frac{1}{2}i + i + 1 \\
&= \frac{1}{2}i^2 + \frac{1}{2}(3i) + \frac{1}{2}(2) \\
&= \frac{1}{2}(i^2 + 3i + 2) \\
&= \frac{1}{2}(i+1)(i+2).
\end{aligned}$$

Note that this final expression is precisely what our claim says the value should be after iteration $i + 1$: total = $\frac{1}{2}(i+1)(i+1+1)$. With this computation done, we can write our argument more formally.

Proof of claim. We argue by induction on i.

Base case. Before the first iteration of the loop, we set total ← 0 in line 2 of ITERATIVESUM. In iteration $i = 1$, line 4 updates the value of total to total + 1 = 0 + 1 = 1. Therefore, total = $1 = \frac{1}{2}(1)(1+1)$ at the end of iteration $i = 1$, so the claim holds for $i = 1$.

Inductive Step. Assume the inductive hypothesis holds—i.e., that after iteration i, total stores the value $\frac{1}{2}i(i+1)$. During iteration $i+1$, line 4 updates the value of total to $\frac{1}{2}i(i+1)+(i+1) = \frac{1}{2}(i+1)(i+2)$ (where the equality holds by the computation we did above). Therefore, the claim holds after iteration $i+1$ as well.

Since we have established that the base case and the inductive step hold, the claim holds by induction. □

Our main proposition now follows immediately from the claim. Specifically, consider the value of total returned by ITERATIVESUM(n). The condition of the for loop in lines 3–5 implies that we break out of the loop after iteration n. By the loop invariant claim, total $= \frac{1}{2}n(n+1)$ after the nth iteration, so this is the value returned by ITERATIVESUM(n) in line 6.

3.2 Recursive Example

Induction is an especially valuable tool in reasoning about recursive methods. For many of us, recursion is an unintuitive way of thinking about computation. Often when one implements a recursive method to solve a problem, it seems to work by magic (if at all). Induction gives us a logical tool to reason about, understand, and justify this magic.

When defining a recursive procedure for a task, we typically design the method in two parts:

- the **base case** in which the procedure should return a value without making a recursive call, and
- the **recursive step** which invokes one or more recursive method calls before returning a value.

In the analysis of a recursively defined method, these two cases correspond to the base case of induction and the inductive step. In the latter case, we can intuitively justify the correctness of the procedure as follows: the a method call succeeds because its recursive calls succeed. The recursive calls succeed because *their* recursive calls succeed, and so on, until a base case is reached. Note that this justification is just applying inductive reasoning in reverse. Once we establish that the base case succeeds and argue the inductive step, we will have established that all recursive method calls succeed. To summarize, *recursion isn't magic, it's induction.*

To give an explicit example, here is a *recursive* implementation of the ITERATIVESUM method defined above:

```
1: procedure RECURSIVESUM(n)          ▷ Sum the numbers from 1 to n
2:     if n ≤ 1 then
3:         return 1
4:     end if
5:     return n + RECURSIVESUM(n − 1)
6: end procedure
```

Exercise 11. *Compute the values returned by* RECURSIVESUM(n) *for* $n = 1,2,3,4$ *by hand to verify that you get the same result as* ITERATIVESUM(n).

We can again use induction to argue that RECURSIVESUM(n) always returns the value $\frac{1}{2}n(n+1)$ for any $n \geq 1$. In this case, the argument is actually a little simpler than the argument for ITERATIVESUM because we do not need to have a separate loop invariant claim. Again, the argument relies on the computation showing that $\frac{1}{2}n(n+1)+(n+1) = \frac{1}{2}(n+1)(n+2)$ that we did before.

Proposition 12. *For every integer $n \geq 1$,* RECURSIVESUM(n) *returns the value $\frac{1}{2}n(n+1)$.*

Proof. We argue the proposition by induction on n.

Base case. In the case $n = 1$, the condition $n \leq 1$ is satisfied in line 2, so the value 1 is returned in line 2. Since $1 = \frac{1}{2}(1)(1+1)$, the proposition holds for $n = 1$.

Inductive step. Suppose the inductive hypothesis holds–i.e., that RECURSIVESUM(n) returns the value $\frac{1}{2}n(n+1)$ for some $n \geq 1$. Since $n+1 > 1$, the value returned by RECURSIVESUM($n+1$) in line 5 is

$$\begin{aligned}(n+1) + \text{RECURSIVESUM}(n) &= (n+1) + \frac{1}{2}n(n+1) \\ &= \frac{1}{2}(n+1)(n+2).\end{aligned}$$

The first equality is from applying the inductive hypothesis, and the second equality holds by the computation we did previously. Therefore, the proposition holds for $n+1$ as well.

Since the base case and inductive step hold, the proposition follows by induction. □