2 LOGICAL AND ALGORITHMIC THINKING

OBJECTIVES

- Learn the importance of logic to computational thinking.
- Appreciate the difference between deductive and inductive reasoning.
- Understand Boolean logic and its importance to computation.
- See the importance of using logical and mathematical notation instead of natural language.
- Learn the properties of algorithms: sequence, iteration, selection.
- Understand the importance of state in algorithms.
- See common mistakes made in logical and algorithmic thinking and learn how to avoid them.

APPROACH

Logic and algorithms are essential to CT. They underpin the subject and rear their heads repeatedly throughout its application. The good news is: humans already have an innate, intuitive understanding of both logic and algorithms. The bad news is: they are both mathematical concepts in nature. Consequently, each has its own set of rules, procedures and definitions, which are very precise and systematic. That means you can't rely solely on your intuition when dealing with these topics, otherwise you'll make mistakes.

The best way to overcome this is to learn the precise, yet difficult core concepts. One could write a whole book just on logic or algorithms (many already have), but such a comprehensive treatment lies far out of scope for this volume, where they are just one topic of many. Consequently, this chapter must narrow its focus.

Research exists that shows us where newcomers tend to make mistakes (Pane et al., 2001), which will allow this chapter to concentrate on the more troublesome spots. I will focus on just a few core elements relevant to getting you into the habit of thinking logically and algorithmically. By the end of the chapter, you will have learned how to apply logic and algorithms to problem-solving. With some practice, they should become second nature.

LOGICAL THINKING

This section shows how important logic is to computational thinking. It introduces logical reasoning, along with Boolean and symbolic logic, and shows how to turn intuitive ideas into mathematically sound, logical expressions.

In brief: Logic

Put simply, logic is a system used for distinguishing between correct and incorrect arguments. By 'argument', I'm not referring to two people in a shouting match. I mean the philosophical idea of an argument; namely a chain of reasoning that ends up in a conclusion.

Logic includes a set of principles that, when applied to arguments, allow us to demonstrate what is true. You need no special training to begin doing this, as this classic introductory example of a logical argument demonstrates:

1. Socrates is a man.
2. All men are mortal.
3. Therefore, Socrates is mortal.

Even those of us without philosophy degrees understand this argument. We perform this particular brand of reasoning all the time (there's a draught in the room; a window is open; therefore, the draught is coming from the window). However, we don't always carry it out **correctly**, which can lead us to form wrong conclusions. And since we use computers essentially to automate our reasoning, we must learn to perform logic correctly before writing a computer solution.

In a sense, applying logic is a way of developing and testing a hypothesis. Using this way of thinking, applying logic assumes you already know at least some things for sure and allows you to use that knowledge to arrive at some further conclusions.

In a logical argument, each individual thing you already know (or assume) is called a **premise**. A premise is like any ordinary statement you or I might make, except that it can be evaluated to obtain an answer of 'true' or 'false'. A premise, therefore, has a **truth value**. In the Socrates example, our premises fit this requirement. It is either true or not that all men are mortal. The same goes for whether Socrates is a man or not. Other forms of expression, like questions or commands, can't be premises to an argument. It's neither true nor false to say 'Break a leg!' or ask 'What time is it?', for example.

Once all the premises are stated, the next step is to analyse them and react accordingly with a conclusion. Most of the magic lies in this step and this is what we will focus on.

Inductive vs deductive arguments

It's important to realise that some logical arguments are stronger than others. In fact, you can categorise arguments based on their certainty. The two best-known categories are **deductive** and **inductive**.

A deductive argument is the strongest form of reasoning because its conclusion necessarily follows from its premises (so long as it has been constructed properly and the premises are incontrovertibly true). We've already seen an example of deductive reasoning: the assessment of Socrates's mortality. While deductive arguments are strong, they have very strict standards, which makes them hard to construct. A deductive argument can fail in one of two ways.[7]

First, one of its premises could turn out to be false. For example:

1. Missie is a dog.
2. All dogs are brown.
3. Therefore, Missie is brown.

Premise 2 is false: not all dogs are brown. Even though the argument follows the exact same form as the Socrates example, it fails because at least one of its premises is false. Any argument with false premises fails. In computer jargon, this is an example of 'garbage in, garbage out'.

You can express the form of an argument by substituting symbols for objects. This helps when trying to work out if your reasoning is valid. The form of the Socrates example is: 'A is a B; All Bs are C; Therefore, A is C.' We'll see later how to handle logic symbolically.

The second way a deductive argument fails is when the conclusion doesn't necessarily follow from the premises. For example:

1. All tennis balls are round.
2. The Earth is round.
3. Therefore, the Earth is a tennis ball.

This argument fails because of faulty logic. Yes, all tennis balls are round, but so are lots of other things. In symbolic terms, this argument follows the form, 'All As are B; C is B; Therefore C is an A', but since this is in an invalid form, the argument is automatically invalid too.

Many books and online resources explain faulty forms of reasoning (i.e. fallacies). A good starting point is *Attacking Faulty Reasoning*, by T. Edward Damer (2005).

In reality, deductive arguments are relatively rare. You'll usually encounter them only when the knowledge you're dealing with is nice and neat. However, the real world often presents us with knowledge that's patchy, messy or provisional. Real-life issues are often nuanced (despite what you might read on social media). For this, we have inductive reasoning, which deals in probabilities rather than hard, black and white rules.

The premises in an inductive argument are not unquestionably true. Rather, we have some level of confidence in them. The form of the argument doesn't guarantee that the conclusion is true, but it probably results in a trustworthy conclusion. For example:

1. A bag contains 99 red balls and one black ball.
2. 100 people each drew one ball from the bag.
3. Sarah is one of those 100 people.
4. Therefore, Sarah probably drew a red ball.

This is a perfectly fine inductive argument, so long as you acknowledge the aspect of probability involved.

These aspects of reasoning are important in CT because computers are involved. The answer a computer gives is only as reliable as its reasoning, and since a computer is automating your reasoning, it's your responsibility to make sure:

1. that reasoning is valid;
2. you give the computer reliable input;
3. you know how to interpret what conclusion the computer reports, that is, is the result unquestionably true (the reasoning was deductive) or probably true (the reasoning was inductive)?

Boolean logic

Even though much of our reasoning is inductive, computers are not well equipped to deal with shades of grey. Their binary nature makes them more apt to deal with black and white issues. In order to instruct computers to make logical decisions, we need a system of logic that maps well onto this way of thinking. Boolean logic is such a method. It's a form of logic that deals with statements having one of only two values: true or false (usually). Different corresponding values could be used in other contexts: 1 or 0 for example, on or off, black or white.

Propositions

Statements in Boolean logic are also known as **propositions**, which have several basic properties.

First, a proposition can only have one value at any one time. In other words, a single proposition can't be both true and false simultaneously. There is no way to express levels of certainty. True means true; false means false. Consequently, you should keep in mind what was said earlier about deductive and inductive arguments: whereas real-life problems often present us with probabilities, the basic Boolean world deals in certainties. Some of your efforts will go into mapping real-world, grey areas onto Boolean black and white.

Second, propositions must have clear and unambiguous meaning. For example, a statement like: 'It is travelling fast', can certainly be evaluated as either true or false. However, it's ambiguous as stated. If 'it' is a car travelling at 150 mph along the

motorway, that's certainly fast. But if 'it' is a spacecraft travelling towards Mars at 150 mph, that's undoubtedly slow.

Qualify statements where necessary to avoid ambiguity. For example, you could restate the example above as: 'It is travelling fast, where "fast" is 70 mph or greater.'

Third, it's possible to combine individual propositions to make more complex ones (called **compound** propositions). For example, 'Jenny is wearing the shirt and the shirt is red.' This is helpful because we often want to evaluate several statements before reaching a conclusion. We make compound propositions by connecting single propositions together using **logical operators**.

Logical operators

Imagine you say, 'If the weather is sunny and I'm on holiday, then I'm going to lie in the garden.' You've just used logical operators. That statement contains two propositions, which serve as conditions for whether, or not, you decide to lounge on the grass:

1. The weather is sunny.
2. You're on holiday.

If both are true, then you can lie in the garden. If **either** of them are false (say, the weather is lousy or you have to go to work), then sunning yourself is not an option. That's because you joined them using the logical operator **AND**, which demands that both propositions should be true for the conclusion to be true.

Many different logical operators exist. It's worth looking at the most important ones in more detail because, even though we use them daily in informal speech, they have specific meanings in logic that occasionally run counter to our intuitive understanding.

To provide illustrative examples, we'll use the operators to describe the rules of a simple game: Noughts and Crosses.[8]

AND: the technical name for this operator is **conjunction**. (We just saw an example of a conjunction when we reasoned about whether to lie in the garden.) It chains propositions together in a way that **all** of them must be true for the conclusion to be true. If any of them are false, the conclusion is rendered false also. In classical logical arguments like we've seen so far, the presence of AND between propositions is implicit, but we can (and should) include them explicitly. So, for example:

1. At least one square on the board is still empty.
2. Neither player has achieved a row.
3. Therefore, the game is still in progress.

This can be expressed as:

> If at least one square on the board is still empty and neither player has achieved a row, then the game is still in progress.

OR: the technical name for this operator is **disjunction**. This operator chains propositions together in a way that **at least one** of them must be true for the conclusion to be true also. The only way that the conclusion is falsified is if all propositions are false. For example:

> If player 1 achieves a row or player 2 achieves a row, then the game is over.

In this case, only one condition needs to be true to end the game. In fact, due to the nature of the game, a maximum of one of these conditions can be true simultaneously. OR can also be used in cases when both conditions can be true at the same time:

> If a player achieves a row or all squares become occupied, then the game is over.

NOT: the technical name for this operator is **negation**. This operator doesn't chain propositions together itself, rather it modifies a single proposition. Specifically, it flips the truth value. Sometimes, negating a proposition can make it easier to express the chain of reasoning. For example:

> If a square is not occupied, then a player may add their symbol to that square.

IMPLIES: the technical name for this operator is **implication**. Using this operator is to state that there is a correlation between the two statements. If the first statement is true, then the second must be true also. Keep in mind, this is a correlation **not** a causation. Therefore, you can't necessarily work backwards from the conclusion of an implication. Consider this:

> If a player achieves a row, then the game is over.

It's true that a game ends when a player achieves a row, but it's not the only way to end a game. Saying 'the game is over, therefore a player achieved a row' is not always true.[9] The game could be over because it ended in a draw.

IF AND ONLY IF: the technical name for this operator is **biconditional**. This behaves very similarly to implication, but a biconditional means that the second proposition is influenced **solely** by the first. If the first is true, the second is true. If the first is false, the second is false. No exceptions. For example:

> If and only if all squares are occupied, then no more moves are possible.

In this case, we **can** work backwards. The only reason no more moves are possible is because all squares are occupied.

Symbolic logic

Logic requires precision and an absence of ambiguity, but it can be difficult to meet these requirements when using natural language. Not only can statements grow wordy and confusing, but their meaning can become almost unavoidably ambiguous.

Saying 'she's a fast reader or she's read the book before' raises the question: could both be true? I would say so. But what about 'Starter is a soup or a salad'? In this case, both can't be true. Notice, however, that in each case we used the word 'or' to join two propositions, but its use implied different meanings in both cases. The other logical operators can similarly be misused to mean different things.

To help us manage our reasoning, mathematics gives us symbolic logic, which recommends using symbols instead of natural language sentences. The earlier example, 'If at least one square on the board is still empty and neither player has achieved a row...' contains two propositions that are replaceable with symbols. If,

P = at least one square on the board is still empty

Q = neither player has achieved a row

S = the game is still in progress

then we can say:

If P and Q, then S

Not only does that reduce the clutter, but it becomes more intuitive to treat each proposition as a variable. After all, each proposition has a value that can be true or false at different times.

The operators, too, are often replaced by symbols (see Table 2.1).

Table 2.1 Logical operators and their symbols

Operator name	Symbol	Example
AND	$\wedge$	$A \wedge B$
OR	$\vee$	$A \vee B$
NOT	$\neg$	$\neg A$
IMPLIES	$\rightarrow$	$A \rightarrow B$
IF AND ONLY IF	$\leftrightarrow$	$A \leftrightarrow B$

Our example can then further be reduced to:

$$P \wedge Q \leftrightarrow S$$

(Rather than overload you with symbols just now, I'll continue to use the operator name rather than the symbol. When you see the operator name in upper case letters, it specifically means the logical operator.)

Symbolic logic also gives each logical operator formal rules, which promote precision and eliminate ambiguity. The meaning of each logical operator is specified in precise mathematical detail. The standard way to do this is to use a truth table. This lists all the possible combinations of values of the propositions. The truth table states whether each combination is logically valid or not. Let's start by looking at the truth table for conjunctions.

To read a truth table, take it one line at a time. The first line of Table 2.2 says that if both propositions are individually valid by themselves, then it's valid to say they are both true together. In the three other eventualities, it is invalid to say they are both true together because one or both propositions are false.

Table 2.2 Truth table of conjunction (logical AND)

P	*Q*	*P* AND *Q*
True	True	True
True	False	False
False	True	False
False	False	False

Keep in mind that the use of 'True' and 'False' can be replaced by other pairs of opposing symbols. Table 2.3 shows the same information as Table 2.2, except that it uses 1 and 0 instead of True and False.

Table 2.3 Truth table of conjunction (using 1 and 0 symbols)

P	*Q*	*P* AND *Q*
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2.1 is a Venn diagram depicting the behaviour of conjunctions.

Figure 2.1 Venn diagram for the AND operator

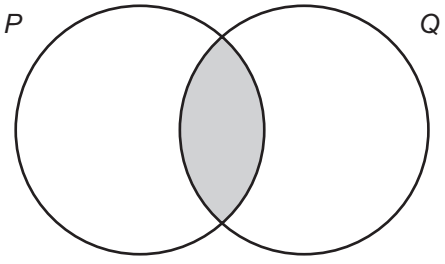

Table 2.4 Truth table of disjunction (logical OR)

P	*Q*	*P* OR *Q*
True	True	True
True	False	True
False	True	True
False	False	False

Table 2.4 shows that it is valid to consider a disjunction true so long as one or more of its propositions are true. That means the intention behind the earlier example – 'A starter can be soup or a salad [but not both]' – doesn't align with this logical meaning of OR (see Figure 2.2).[10]

Table 2.5 Truth table of negation (logical NOT)

P	NOT *P*
True	False
False	True

Figure 2.2 Venn diagram for the OR operator

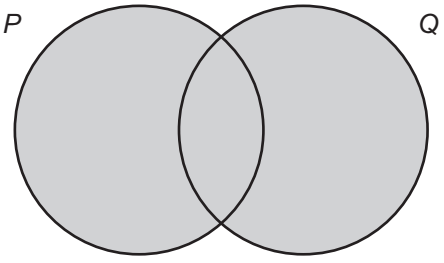

Figure 2.3 pictorially represents the difference between a proposition with and without the NOT operator. In other words, whatever is true becomes false, and whatever is false becomes true.

Figure 2.3 Venn diagram for the NOT operator

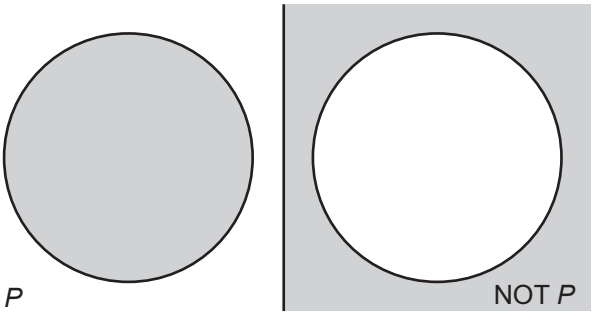

Table 2.6 Truth table of implication (logical IMPLIES)

P	*Q*	*P* IMPLIES *Q*
True	True	True
True	False	False
False	True	True
False	False	True

The first two lines of Table 2.6 seem sensible. If *P* being true implies *Q* is true and both are indeed true, then the implication is valid. Likewise, if *P* is true, but its consequent is not true, then the implication is invalid. The last two lines, however, seem odd. The implication is valid regardless of the value of *P*.

Why is it valid for *P* to imply *Q* when *P* is false? In '*P* implies *Q*', *P* is known as the antecedent and *Q* as the consequent. The explanation is that no conclusion can be drawn about an implication when the antecedent (i.e. *P*) is false. Consider the statement 'if it is raining (*P*), then the grass is wet (*Q*)'. Saying 'it is not raining' (NOT *P*) doesn't contradict that the grass is wet. The grass could be wet for another reason, such as because the sprinkler is on. Since there is no contradiction, mathematicians have judged that an implication with a false antecedent is true until proven otherwise (see Figure 2.4).

Figure 2.4 Venn diagram for the IMPLIES operator

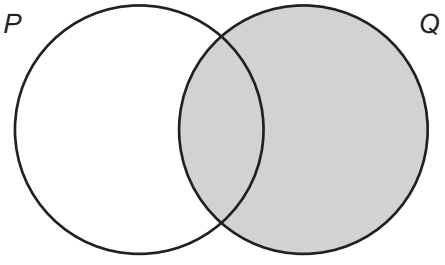

Table 2.7 Truth table of biconditional (logical IF AND ONLY IF)

P	*Q*	*Q* IF AND ONLY IF *P*
True	True	True
True	False	False
False	True	False
False	False	True

Figure 2.5 depicts the IF AND ONLY IF operator. Only when values agree can the statement be considered valid. *P* and *Q* can agree when their values overlap, either by being both true or both false.

Figure 2.5 Venn diagram for the IF AND ONLY IF operator