Chapter 2

Propositional Logic

Overview The most basic logical inferences are about combinations of sentences, expressed by such frequent expressions as ‘not’, ‘and’, ‘or’, ‘if, then’. Such combinations allow you to describe situations, and what properties these situations have or lack: something is ‘not this, but that’. You could call this reasoning about ‘classification’, and it is the basis of any description of the world. At the same time, these logical sentence combinations are also fundamental in another sense, as they structure how we communicate and engage in argumentation. When you disagree with a claim that someone makes, you often try to derive a consequence (’if then’) whose negation (‘not’) is easier to show. We will study all these patterns of reasoning below.

More precisely, in this first chapter you will be introduced to *propositional logic*, the logical system behind the reasoning with ‘not’, ‘and’, ‘or’, ‘if, then’ and other basic sentence-combining operators. You will get acquainted with the notions of formula, logical connective, truth, valid consequence, information update, formal proof, and expressive power, while we also present some backgrounds in computation and cognition.

2.1 Reasoning in Daily Life

Logic can be seen in action all around us:

> In a restaurant, your Father has ordered Fish, your Mother ordered Vegetarian, and you ordered Meat. Out of the kitchen comes some new person carrying the three plates. What will happen?

We have know this from experience. The waiter asks a first question, say “Who ordered the meat?”, and puts that plate. Then he asks a second question “Who has the fish?”, and puts that plate. And then, without asking further, he knows he has to put the remaining plate in front of your Mother. What has happened here?

Starting at the end, when the waiter puts the third plate without asking, you see a major logical act ‘in broad daylight’: the waiter draws a conclusion. The information in the two answers received allows the waiter to infer automatically where the third dish must go. We represent this in an *inference schema* with some special notation (F for “fish”, M for “meat”, V for “vegetarian”):

$$F \text{ or } V \text{ or } M, \text{not } M, \text{ not } F \Longrightarrow V. \tag{2.1}$$

This formal view has many benefits: one schema stands for a wide range of inferences, for it does not matter what we put for F, V and M.

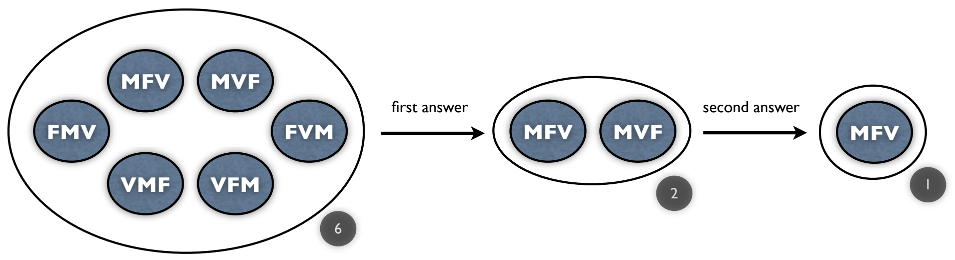
Inferences often come to the surface especially vividly in puzzles, where we exercise our logical abilities just for the fun of it. Think of successive stages in the solution of a 3×3 Sudoku puzzle, produced by applying the two basic rules that each of the 9 positions must have a digit, but no digit occurs twice on a row or column:

$$\boxed{\begin{array}{ccc} 1 & \bullet & \bullet \\ \bullet & \bullet & 2 \\ \bullet & \bullet & \bullet \end{array}}\ , \quad \boxed{\begin{array}{ccc} 1 & \bullet & 3 \\ \bullet & \bullet & 2 \\ \bullet & \bullet & \bullet \end{array}}\ , \quad \boxed{\begin{array}{ccc} 1 & 2 & 3 \\ \bullet & \bullet & 2 \\ \bullet & \bullet & \bullet \end{array}} \quad \cdots \tag{2.2}$$

Each successive diagram displays a bit more explicit information about the solution, which is already implicitly determined by the initial placement of the two digits 1, 2. And the driving mechanism for these steps is exactly our Restaurant inference. Think of the step from the first to the second picture. The top right dot is either 1, 2 or 3. It is not 1. It is not 2. Therefore, it has to be 3.

But is much more information flow in this Restaurant scene. Before his final inference, the waiter first actively sought to find out enough facts by another typical information-producing act, viz. asking a question. And the answers to his two questions were also crucial.

The essence of this second process is a form of computation on information states. During a conversation, information states of people – singly, and in groups – change over time, triggered by communicative events. The Restaurant scenario starts with an initial information state consisting of six options, all the ways in which three plates can be distributed over three people $(MFV, MVF, ...)$. The answer to the first question then reduces this to two (the remaining orders FV, VF), and the answer to the second question reduces this to one, zooming in on just the actual situation (for convenience, assume it is MFV). This may be pictured as a diagram (‘video’) of successive updates:



(2.3)

2.2 Inference Patterns, Validity, and Invalidity

Consider the following statement from your doctor:

If you take my medication, you will get better.
But you are not taking my medication.

So, you will not get better. (2.4)

Here the word 'so' (or 'therefore', 'thus', etc.) suggests the drawing of a conclusion from two pieces of information: traditionally called the 'premises'. We call this an act of inference. Now, as it happens, this particular inference is not compelling. The conclusion might be false even though the two premises are true. You might get better by taking that greatest medicine of all (but so hard to swallow for modern people): just wait. Relying on a pattern like this might even be pretty dangerous in some scenarios:

If I resist, the enemy will kill me.
But I am not resisting.

So, the enemy will not kill me. (2.5)

Now contrast this with another pattern:

If you take my medication, you will get better.
But you are not getting better.

So, you have not taken my medication. (2.6)

This is valid: there is no way that the two stated premises can be true while the conclusion is false. It is time for a definition. Broadly speaking, we call an inference *valid* if there is 'transmission of truth': in every situation where all the premises are true, the conclusion is also true.

Stated differently but equivalently, an inference is valid if it has no 'counter-examples': that is, situations where the premises are all true while the conclusion is false. This is a crucial notion to understand, so we dwell on it a bit longer.

What validity really tells us While this definition makes intuitive sense, it is good to realize that it may be weaker than it looks a first sight. For instance, a valid inference with two premises

$$P_1, P_2, \text{ so } C \tag{2.7}$$

allows many combinations of truth and falsity. If any premise is false, nothing follows about the conclusion. In particular, in the second doctor example, the rule may hold (the first premise is true), but you are getting better (false second premise), and you did take the medication (false conclusion). Of all eight true-false combinations for three sentences, validity rules out 1 (true-true-false)! The most you can say for sure thanks to the validity can be stated in one of two ways:

$$\begin{array}{l}\text{(a) if all premises are true, then the conclusion is true}\\ \text{(b) if the conclusion is false, then at least one premise is false}\end{array} \tag{2.8}$$

The first version is how people often think of logic: adding more things that you have to accept given what you have accepted already. But there is an equally important use of logic in *refuting* assertions, perhaps made by your opponents. You show that some false consequence follows, and then cast doubt on the original assertion. The second formulation says exactly how this works. Logical inferences also help us see what things are false — or maybe more satisfyingly, refute someone else. But note the subtlety: a false conclusion does not mean that all premises were false, just that at least one is. Detecting this bad apple in a basket may still take further effort.

To help you understand both aspects of validity, consider the tree below: representing a 'complex argument' consisting of individual inferences with capital letters for sentences, premises above the bar, and the conclusion below it. Each inference in the tree is valid:

$$\frac{\dfrac{\mathsf{A} \qquad \dfrac{\mathsf{A}\ \ \mathsf{B}\ \ \mathsf{C}}{\mathsf{D}}}{\mathsf{E}} \qquad \dfrac{\mathsf{E}}{\mathsf{F}} \qquad \dfrac{\mathsf{A}}{\mathsf{B}}}{\mathsf{G}} \tag{2.9}$$

You are told reliably that sentence A is true and G is false. For which further sentences that occur in the tree can you now determine their truth and falsity? (The answer is that A, B, are true, C, D, E , G are false, while we cannot tell whether F is true or false.)

Inference patterns The next step in the birth of logic was the insight that the validity and invalidity here have to do with abstract patterns, the shapes of the inferences, rather than their specific content. Clearly, the valid second argument would also be valid in the following concrete form, far removed from doctors and medicine:

$$\begin{array}{l} \text{If the enemy cuts the dikes, Holland will be inundated.} \\ \text{Holland is not inundated.} \\ \hline \text{So, the enemy has not cut the dikes.} \end{array} \qquad (2.10)$$

This form has variable parts (we have replaced some sentences by others), but there are also constant parts, whose meaning must stay the same, if the inference is to be valid. For instance, if we also replace the negative word ‘not’ by the positive word ‘indeed’, then we get the clearly invalid inference:

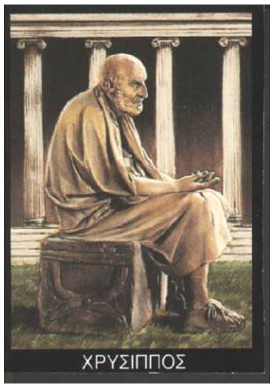
$$\begin{array}{l} \text{If the enemy cuts the dikes, Holland will be inundated.} \\ \text{Holland is indeed inundated.} \\ \hline \text{So, the enemy has indeed cut the dikes.} \end{array} \qquad (2.11)$$

For counter-examples: the inundation may be due to faulty water management, rain, etc.

To bring out the relevant shared underlying form of inferences, we need a notation for both fixed and variable parts. We do this by using variable letters for expressions that can be replaced by others in their linguistic category, plus special notation for key expressions that determine the inference, often called the logical constants.

2.3 Classification, Consequence, and Update

Classification The main ideas of propositional logic go back to Antiquity (the Stoic philosopher Chrysippus of Soli, c.280–c.207 BC), but its modern version starts in the nineteenth century, with the work of the British mathematician George Boole (1815–1864).



Chrysippus

George Boole

Our earlier examples were essentially about combinations of propositions (assertions expressed by whole sentences). From now on, we will indicate basic propositions by letters $p, q,$ etcetera. A finite number of such propositions generates a finite set of possibilities, depending on which are true and which are false. For instance, with just p, q there are four true/false combinations, that we can write as

$$pq, p\overline{q}, \overline{p}q, \overline{pq} \tag{2.12}$$

where p represents that p is true and $\overline{p}$ that p is false. Thus, we are interested in a basic logic of this sort of classification. (Note that $\overline{p}$ is not meant as a logical proposition here, so that it is different from the negation not-p that occurs in inferences that we will use below. The distinction will only become clear later.)

Drawing consequences Now consider our earlier examples of valid and invalid arguments. For instance,

(a) the argument “from if-p-then-q and not-p to not-q” was invalid,

whereas

(b) the argument “from if-p-then-q, not-q to not-p” was valid.

Our earlier explanation of validity for a logical consequence can now be sharpened up. In this setting, it essentially says the following:

> each of the above four combinations that makes the premises true must also make the conclusion true.

You can check whether this holds by considering all cases in the relevant list that satisfy the premises. For instance, in the first case mentioned above,

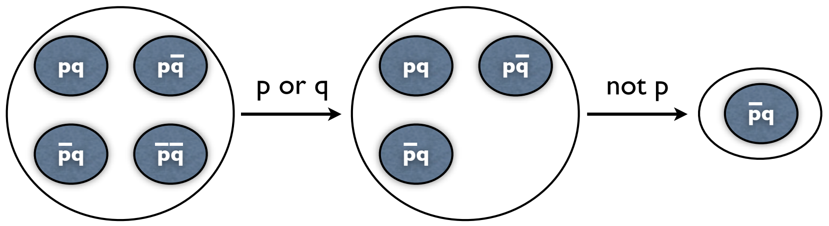
(a) not-p is true at $\overline{p}q$ and $\overline{pq}$. if-p-then-q holds also in these two situations, since the condition p is not true. So, the first of the two situations, $\overline{p}q$, support the two premises but the conclusion not-q is false in this situation. The argument is therefore invalid!

For the second case we get

(b) not-q is true at $p\overline{q}$ and $\overline{pq}$. while if-p-then-q only holds in the second, so $\overline{pq}$ is the only situation in which all the premises hold. In this situation not-p also holds, and therefore, the argument is valid.

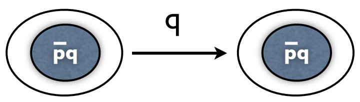
Updating information Propositional logic describes valid (and invalid) inference patterns, but it also has other important uses. In particular, it describes the information flow in earlier examples, that may arise from observation, or just facts that are being told.

With the set of all combinations present, we have no information about the actual situation. But we may get additional information, ruling out options. To see how, consider a simple party, with just two possible invitees Mary and John. We write p and q, respectively, for "Mary comes to the party" and "John comes to the party". Suppose that we are first told that at least one of the invitees comes to the party: p-or-q. Out of four possible situations this new information rules out just one, viz. $\overline{pq}$. Next, the we learn that not-p. This rules out two more options, and we are left with only the actual situation $\overline{p}q$. Here is a 'video-clip' of the successive information states, that get 'updated' by new information:



(2.13)

Incidentally, you can now also see why the conclusion q is a valid inference from 'p or q' and 'not p'. Adding the information that q does not change the final information state, nothing is ruled out:



(2.14)

But if adding the information that q does not change anything, this means that q is already true. So the truth of q is guaranteed by the fact that the two earlier updates have taken place. This must mean that q is logically implied by the earlier formulas.

Exercise 2.1 Consider the case where there are three facts that you are interested in. You wake up, you open your eyes, and you ask yourself three things: "Have I overslept?", "Is it raining?", "Are there traffic jams on the road to work?". To find out about the first question, you have to check your alarm clock, to find out about the second you have to look out of the window, and to find out about the third you have to listen to the traffic info on the radio. We can represent these possible facts with three basic propositions, p, q and r, with p expressing "I have overslept", q expressing "It is raining", and r expressing "There are traffic jams." Suppose you know nothing yet about the truth of your three facts. What is the space of possibilities?

Exercise 2.2 (Continued from previous exercise.) Now you check your alarm clock, and find out that you have not overslept. What happens to your space of possibilities?

Toward a system Once we have a system in place for these tasks, we can do many further things. For instance, instead of asking whether a given inference is valid, we can also just look at given premises, and ask what would be a most informative conclusion. Here is a case that you can think about (it is used as a basic inference step to program computers that perform reasoning automatically):

Exercise 2.3 You are given the information that p-or-q and (not-p)-or-r. What can you conclude about q and r? What is the strongest valid conclusion you can draw? (A statement is stronger than another statement if it rules out more possibilities.)

A precise system for the above tasks can also be *automated*, and indeed, propositional logic is historically important also for its links with computation and computers. Computers become essential with complex reasoning tasks, that require many steps of inference or update of the above simple kinds, and logical systems are close to automated deduction. But as we shall see later in Section 2.10, there is even a sense in which propositional logic *is* the language of computation, and it is tied up with deep open problems about the nature of computational complexity.

But the start of our story is not in computation, but in natural language. We will identify the basic expressions that we need, and then sharpen them up in a precise notation.

2.4 The Language of Propositional Logic

Reasoning about situations involves complex sentences with the 'logical connectives' of natural language, such as 'not', 'and', 'or' and 'if .. then'. These are not the only expressions that drive logical reasoning, but they do form the most basic level. We could stay close to natural language itself to define our system (traditional logicians often did), but it has become clear over time that working with well-chosen notation makes things much clearer, and easier to manipulate. So, just like mathematicians, logicians use formal notations to improve understanding and facilitate computation.

From natural language to logical notation As we have seen in Section 2.3, logical forms lie behind the valid inferences that we see around us in natural language. So we need a good notation to bring them out. For a start, we will use special symbols for the key logical operator words:

Symbol	In natural language	Technical name
$\neg$	not	negation
$\wedge$	and	conjunction
$\vee$	or	disjunction
$\to$	if ... then	implication
$\leftrightarrow$	if and only if	equivalence

(2.15)

Other notations occur in the literature, too: some dialects have & for $\wedge$, and $\equiv$ for $\leftrightarrow$. We write small letters for basic propositions p, q, etcetera. For arbitrary propositions, which may contain connectives as given in the table (2.15), we write small Greek letters φ, ψ, χ, etc.

Inclusive and exclusive disjunction The symbol $\vee$ is for inclusive disjunction, as in 'in order to pass the exam, question 3 or question 4 must have been answered correctly'. Clearly, you don't want to be penalized if both are correct! This is different from the *exclusive disjunction* (most often written as $\oplus$), as in 'you can marry Snowwhite or Cinderella'. This is not an invitation to marry both at the same time. When we use the word 'disjunction' without further addition we mean the inclusive disjunction.

Now we can write logical forms for given assertions, as 'formulas' with these symbols. Consider a card player describing the hand of her opponent:

Sentence	"He has an Ace if he does not have a Knight or a Spade"
Logical formula	$\neg(k \vee s) \to a$

It is useful to see this process of formalization as something that is performed in separate steps, for example, as follows. In cases where you are in doubt about the formalization of a phrase in natural language, you can always decide to 'slow down' to such a stepwise analysis, to find out where the crucial formalization decision is made.

He has an Ace if he does not have a Knight or a Spade,
if (he does not have a Knight or a Spade), then (he has an Ace),
(he does not have a Knight or a Spade) $\to$ (he has an Ace),
not (he has a Knight or a Spade) $\to$ (he has an Ace),
$\neg$ (he has a Knight or a Spade) $\to$ (he has an Ace),
$\neg$ ((he has a Knight) or (he has a Spade)) $\to$ (he has an Ace),
$\neg$ ((he has a Knight) $\vee$ (he has a Spade)) $\to$ (he has an Ace),
$\neg(k \vee s) \to a$

In practice, one often also sees mixed notations where parts of sentences are kept intact, with just logical keywords in formal notation. This is like standard mathematical language, that mixes symbols with natural language. While this mixing can be very useful (the notation enriches the natural language, and may then be easier to absorb in cognitive practice), you will learn more here by looking at the extreme case where the whole sentence is replaced by a logical form.

Ambiguity The above process of taking natural language to logical forms is not a routine matter. There can be quite some slack, with genuine issues of interpretation. In particular, natural language sentences can be *ambiguous*, having different interpretations. For instance, another possible logical form for the card player's assertion is the formula

$$(\neg k \vee s) \to a \tag{2.16}$$

Check for yourself that this says something different from the above. One virtue of logical notation is that we see such differences at a glance: in this case, by the placement of the brackets, which are auxiliary devices that do not occur as such in natural language (though it has been claimed that some actual forms of expression do have 'bracketing functions').

Sometimes, the logical form of what is stated is even controversial. According to some people, 'You will get a slap (s), unless you stop whining ($\neg w$)' expresses the implication $w \to s$. According to others, it expresses the equivalence $w \leftrightarrow s$. Especially, negations in natural language may quickly get hard to grasp. Here is a famous test question in a psychological experiment that many people have difficulty with. How many negations are there, and what does the stacking of negations mean in the following sentence:

> "Nothing is too trivial to be ignored?"

Formal language and syntactic trees Logicians think of the preceding notations, not just as a device that can be inserted to make natural language more precise, but as something that is important on its own, namely, an artificial or formal language.

You can think of formulas in such a language as being constructed, starting from basic propositions, often indicated by letters p, q, etcetera, and then applying logical operations, with brackets added to secure unambiguous readability.

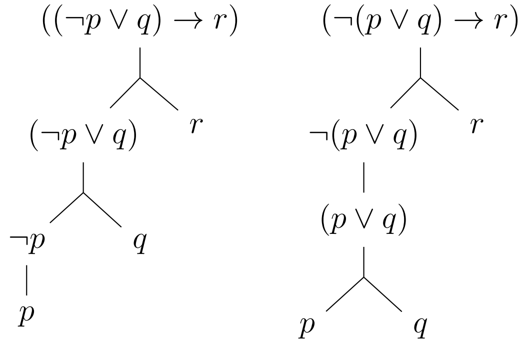
Example 2.4 The formula $((\neg p \vee q) \to r)$ is created stepwise from proposition letters p, q, r by applying the following construction rules successively:

(a) from p, create $\neg p$,

(b) from $\neg p$ and q, create $(\neg p \vee q)$

(c) from $(\neg p \vee q)$ and r, create $((\neg p \vee q) \to r)$

This construction may be visualized in *trees* that are completely unambiguous. Here are trees for the preceding example plus a variant that we already noted above. Mathematically, bracket notation and tree notation are equivalent — but their cognitive appeal differs, and trees are widely popular in mathematics, linguistics, and elsewhere:



This example has prepared us for the formal presentation of the language of propositional logic. There are two ways to go about this, they amount to the same: an 'inductive definition' (for this technical notion, see Appendix A). Here is one way:

> Every proposition letter $(p, q, r, \ldots)$ is a formula. If φ is a formula, then $\neg\varphi$ is also a formula. If φ_1 and φ_2 are formulas, then $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \to \varphi_2)$ and $(\varphi_1 \leftrightarrow \varphi_2)$ are also formulas. Nothing else is a formula.

We can now clearly recognize that the way we have constructed the formula in the example above is exactly according to this pattern. That is merely a particular instance of the above definition. The definition is formulated in more abstract terms, using the formula variables φ_1 and φ_2. An even more abstract specification, but one that amounts to exactly the same inductive definition, is the so-called BNF specification of the language of propositional logic. BNF stands for 'Backus Naur Form', after the computer scientists John Backus and Peter Naur who introduced this device for the syntax of programming languages.

Definition 2.5 (Language of propositional logic) Let P be a set of proposition letters and let $p \in P$.

$$\varphi \quad ::= \quad p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \to \varphi) \mid (\varphi \leftrightarrow \varphi)$$

We should read such a definition as follows. In the definition we define objects of the type 'formula in propositional logic', in short: formulas. The definition starts by stating that every atomic proposition is of that type, i.e., is a formula. Then it says that if an object is of type φ, then $\neg\varphi$ is also of type φ. Note that it does not say that $\neg\varphi$ is the same formula φ. It merely says that both can be called 'formula'. This definition then helps us to construct concrete formulas step by step, as in the previous example.

Backus Naur form is an example of linguistic specification. In fact, BNF is a computer science re-invention of a way to specify languages that was proposed in 1956 by the linguist Noam Chomsky.

In practice we often do not write the parentheses, and we only keep them if their removal would make the expression ambiguous, as in $p \vee q \wedge r$. This can mean $((p \vee q) \wedge r)$ but also $(p \vee (q \wedge r))$ and that makes quite a difference. The latter is already true if only p is true, but the former requires r to be true. Or take a natural language example: "Haddock stays sober or he drinks and he gets angry."

Exercise 2.6 Write in propositional logic:

- I will only go to school if I get a cookie now.
- John and Mary are running.
- A foreign national is entitled to social security if he has legal employment or if he has had such less than three years ago, unless he is currently also employed abroad.

Exercise 2.7 Which of the following are formulas in propositional logic:

- $p \to \neg q$
- $\neg\neg \wedge q \vee p$
- $p\neg q$

Exercise 2.8 Construct trees for the following formulas:

- $(p \wedge q) \to \neg q$
- $q \wedge r \wedge s \wedge t$ (draw all possible trees: think of bracket arrangements).

Exercise 2.9 From the fact that several trees are possible for $q \wedge r \wedge s \wedge t$, we see that this expression can be read in more than one way. Is this ambiguity harmful or not? Why (not)? If you find this hard to answer, think of a natural language example.

A crucial notion: pure syntax Formulas and trees are pure symbolic forms, living at the level of syntax, as yet without concrete meaning. Historically, identifying this separate level of form has been a major abstraction step, that only became fully clear in 19th century mathematics. Most uses of natural language sentences and actual reasoning come with meanings attached, unless very late at parties. Pure syntax has become the basis for many connections between logic, mathematics, and computer science, where purely symbolic processes play an important role.

Logic, language, computation, and thought The above pictures may remind you of parse trees in grammars for natural languages. Indeed, translations between logical forms and linguistic forms are a key topic at the interface of logic and linguistics, which has also started working extensively with mathematical forms in the 20th century. Connections between logical languages and natural language have become important in Computational Linguistics and Artificial Intelligence, for instance when interfacing humans with computers and symbolic computer languages. In fact, you can view our syntax trees in two ways, corresponding to two major tasks in these areas. 'Top down' they analyze complex expressions into progressively simpler ones: a process of *parsing* given sentences. But 'bottom up' they construct new sentences, a task called language generation.

But also philosophically, the relation between natural and artificial languages has been long under debate. The more abstract level of logical form has been considered more 'universal' as a sort of 'language of thought', that transcends differences between natural languages (and perhaps even between cultures). You can also cast the relation as a case of replacement of messy ambiguous natural language forms by clean logical forms for reasoning and perhaps other purposes — which is what the founding fathers of modern logic had in mind, who claimed that natural languages are 'systematically misleading'. But less radically, and perhaps more realistic from an empirical cognitive viewpoint, you can also see the relation as a way of creating *hybrids* of existing and newly designed forms of expression. Compare the way the language of mathematicians consists of natural language plus a growing fund of notations, or the way in which computer science extends our natural repertoire of expression and communication.

2.5 Semantic Situations, Truth Tables, Binary Arithmetic

Differences in formal syntax often correspond to differences in meaning: the above two trees are an example. To explain this in more detail, we now need a semantics that, for a start, relates syntactic objects like formulas to truth and falsity in semantic situations. Thus, formulas acquire meaning in specific settings, and differences in meaning between formulas are often signalled by differences in truth in some situation.

Truth values and valuations for atoms As we said already, each set of proposition letters $p, q, r, \ldots$ generates a set of different *situations*, different ways the actual world might be, or different states that it could be in (all these interpretations make sense in applications). Three proposition letters generate $2^3 = 8$ situations:

$$\{pqr, pq\overline{r}, p\overline{q}r, p\overline{qr}, \overline{p}qr, \overline{p}q\overline{r}, \overline{pq}r, \overline{pqr}\} \tag{2.17}$$

Here proposition letters stand for 'atomic propositions', while logical operations form 'molecules'. Of course this is just a manner of speaking, since what counts as 'atomic' in a given application is usually just our decision 'not to look any further inside' the proposition. A convenient mathematical view of situations is as functions from atomic propositions to truth values 1 ('true') and 0 ('false'). For instance, the above situation $p\overline{q}r$ corresponds to the function sending p to 1, q to 0, and r to 1. An alternative notation for truth values is t and f, but we use numbers for their suggestive analogy with binary arithmetic (the heart of computers). We call these functions V *valuations*; $V(\varphi) = 1$ says that the formula φ is true in the situation (represented by) V, and $V(\varphi) = 0$ says that the formula φ is false in the situation V. For $V(\varphi) = 1$ we also write $V \models \varphi$ and for $V(\varphi) = 0$ we also write $V \not\models \varphi$. One can read $V \models \varphi$ as "V makes true φ", or as "V *satisfies* φ" or "V is a *model* of φ". The notation using $\models$ will reappear in later chapters.

Boolean operations on truth values Any complex sentence constructed from the relevant atomic proposition letters is either true or false in each situation. To see how this works, we first need an account for the meaning of the logical operations. This is achieved by assigning them Boolean operations on the numbers 0, 1, in a way that respects (as far as reasonable) their intuitive usage. For instance, if $V(\varphi) = 0$, then $V(\neg\varphi) = 1$, and vice versa; and if $V(\varphi) = 1$, then $V(\neg\varphi) = 0$, and vice versa. Such relations are easier formatted in a table.

Definition 2.10 (Semantics of propositional logic) A valuation V is a function from proposition letters to truth values 0 and 1. The value or meaning of complex sentences is computed from the value of basic propositions according to the following truth tables.

φ	$\neg\varphi$
0	**1**
1	**0**

φ	ψ	$\varphi \wedge \psi$	$\varphi \vee \psi$	$\varphi \to \psi$	$\varphi \leftrightarrow \psi$
0	0	**0**	**0**	**1**	**1**
0	1	**0**	**1**	**1**	**0**
1	0	**0**	**1**	**0**	**0**
1	1	**1**	**1**	**1**	**1**

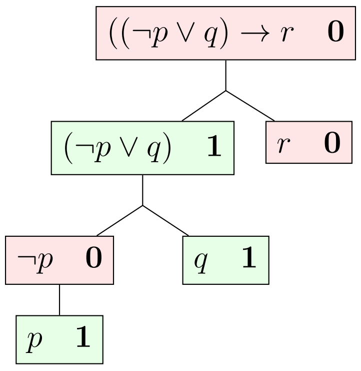
(2.18)

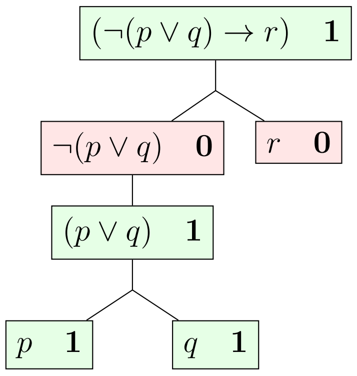
Bold-face numbers give the truth values for all relevant combinations of argument values: four in the case of connectives with two arguments, two in the case of the connective with one argument, the negation.

Explanation The tables for negation, conjunction, disjunction, and equivalence are quite intuitive, but the same does not hold for the table for implication. The table for implication has generated perennial debate, since it does not match the word 'implies' in natural language very well. E.g., does having a false *antecedent* (condition) φ and a true *consequent* ψ really make the implication if-φ-then-ψ true? But we are just doing the best we can in our simple two-valued setting. Here is a thought that has helped many students. You will certainly accept the following assertion as true: 'All numbers greater than 13 are greater than 12'. Put differently, 'if a number n is greater than 13 (p), then n is greater than 12 (q)'. But now, just fill in different numbers n, and you get all combinations in the truth table. For instance, $n = 14$ motivates the truth-value 1 for $p \to q$ at pq, $n = 13$ motivates 1 for $p \to q$ at $\overline{p}q$, and $n = 12$ motivates 1 for $p \to q$ at $\overline{pq}$.

A mismatch with natural language can actually be very useful. Conditionals are a 'hot spot' in logic, and it is a challenge to create systems that get closer to their behaviour. Propositional logic is the simplest treatment that exists, but other logical systems today deal with further aspects of conditionals in natural language and ordinary reasoning. You will see a few examples later in this course.

Computing truth tables for complex formulas How exactly can we compute truth values for complex formulas? This is done using our tables by following the construction stages of syntax trees. Here is how this works. Take the valuation V with $V(p) = V(q) = 1, V(r) = 0$ and consider two earlier formulas:





Incidentally, this difference in truth value explains our earlier point that these two variant formulas are different readings of the earlier natural language sentence.

Computing in this manner for all valuations, we can systematically tabulate the truth value behaviour of complex propositional formulas in all relevant situations:

p	q	r	$((\neg p \lor q) \to r)$	$(\neg(p \lor q) \to r)$
0	0	0	**0**	**0**
0	0	1	**1**	**1**
0	1	0	**0**	**1**
0	1	1	**1**	**1**
1	0	0	**1**	**1**
1	0	1	**1**	**1**
1	1	0	**0**	**1**
1	1	1	**1**	**1**

(2.19)

Paying attention to the proper placement of brackets in formulas, you can compute truth-tables step by step for all situations. As an example we take the second formula from (2.19). First, start with summing up the situations and copy the truth-values under the proposition letters as has been done in the following table.

p	q	r	$(\neg$	$(p$	$\lor$	$q)$	$\to$	$r)$
0	0	0	·	0	·	0	·	0
0	0	1	·	0	·	0	·	1
0	1	0	·	0	·	1	·	0
0	1	1	·	0	·	1	·	1
1	0	0	·	1	·	0	·	0
1	0	1	·	1	·	1	·	1
1	1	0	·	1	·	0	·	0
1	1	1	·	1	·	1	·	1

(2.20)

Then start filling in the truth-values for the first possible operator. Here it is the disjunction: it can be computed because the values of its arguments are given (you can also see this from the construction tree). $(p \lor q)$ gets value 0 if and only if both p and q have the value 0. The intermediate result is given in the first table in (2.21). The next steps are the negation and then the conjunction. This gives the following results:

$(\neg$	$(p$	$\vee$	$q)$	$\to$	$r)$
·	0	0	0	·	0
·	0	0	0	·	1
·	0	1	1	·	0
·	0	1	1	·	1
·	1	1	0	·	0
·	1	1	0	·	1
·	1	1	1	·	0
·	1	1	1	·	1

$(\neg$	$(p$	$\vee$	$q)$	$\to$	$r)$
1	0	0	0	·	0
1	0	0	0	·	1
0	0	1	1	·	0
0	0	1	1	·	1
0	1	1	0	·	0
0	1	1	0	·	1
0	1	1	1	·	0
0	1	1	1	·	1

$(\neg$	$(p$	$\vee$	$q)$	$\to$	$r)$
1	0	0	0	**0**	0
1	0	0	0	**1**	1
0	0	1	1	**1**	0
0	0	1	1	**1**	1
0	1	1	0	**1**	0
0	1	1	0	**1**	1
0	1	1	1	**1**	0
0	1	1	1	**1**	1

(2.21)

One does not have to draw three separate tables. All the work can be done in a single table. We just meant to indicate the right order of filling in truth-values.

Exercise 2.11 Construct truth tables for the following formulas:

- $(p \to q) \vee (q \to p)$,
- $((p \vee \neg q) \wedge r) \leftrightarrow (\neg(p \wedge r) \vee q)$.

Exercise 2.12 Using truth tables, investigate all formulas that can be readings of

$$\neg p \to q \vee r$$

(by inserting brackets in appropriate places), and show that they are not equivalent.

If, Only If, If and Only If Here is a useful list of different ways to express implications:

If p then q	$p \to q$
p if q	$q \to p$
p only if q	$p \to q$

The third item on this list may come as a surprise. To see that the third item is correct, reflect on how one can check whether “We will help you only if you help us” is false. This can can happen only if “We help you” is true, but “You help us” is false.

These uses of ‘if’ and ‘only if’ explain the use of the common abbreviation ‘if and only if’ for an equivalence. “We will help you if and only if you help us” states that “you help us” implies “we help you”, and vice versa. A common abbreviation for ’if and only if’ that we will use occasionally is *iff*.

2.6 Valid Consequence and Consistency

We now define the general notion of valid consequence for propositional logic. It is a more precise version of the notion of a valid argument that we introduced on page 2-4.

The notion runs over all possible valuations, and as we will see in a moment, we can use truth tables to check given inferences for validity. (In what follows, k can be any number. If it is $k = 0$, then there are no premises.)

Definition 2.13 (Valid consequence) The inference from a finite set of premises

$$\varphi_1, \ldots, \varphi_k$$

to a conclusion ψ is a *valid consequence*, something for which we write

$$\varphi_1, \ldots, \varphi_k \models \psi,$$

if each valuation V with $V(\varphi_1) = \ldots = V(\varphi_k) = 1$ also has $V(\psi) = 1$.

Definition 2.14 (Logical equivalence) If $\varphi \models \psi$ and $\psi \models \varphi$ we say that φ and ψ are *logically equivalent*.

Here it is useful to recall a warning that was already stated above. Do not confuse valid consequence with truth of formulas in a given situation: validity quantifies over truth in many situations, but it has no specific claim about truth or falsity of the premises and conclusions in the situation. Indeed, validity rules out surprisingly little in this respect: of all the possible truth/falsity combinations that might occur for premises and conclusion, it only rules out one case: viz. that all φ_i get value 1, while ψ gets value 0.

Another point from Section 2.2 that is worth repeating here concerns the role of propositional inference in conversation and argumentation. Valid inference does not just help establish truth, but it can also achieve a refutation of claims: when the conclusion of a valid consequence is false, at least one of the premises must be false. But logic does not tell us in general which one: some further investigation may be required to find the culprit(s). It has been said by philosophers that this refutational use of logic may be the most important one, since it is the basis of *learning*, where we constantly have to give up current beliefs when they contradict new facts.

Here is a simple example of how truth tables can check for validity:

Example 2.15 (Modus Tollens) The simplest case of refutation depends on the rule of *modus tollens*:

$$\varphi \to \psi, \neg\psi \models \neg\varphi.$$

Below you see the complete truth table demonstrating its validity:

φ	ψ	$\varphi \to \psi$	$\neg\psi$	$\neg\varphi$	
1	1	**1**	**0**	**0**	
1	0	**0**	**1**	**0**	
0	1	**1**	**0**	**1**	
0	0	**1**	**1**	**1**	!!

(2.22)

Of the four possible relevant situations here, only one satisfies both premises (the valuation on the fourth line), and we can check that there, indeed, the conclusion is true as well. Thus, the inference is valid.

By contrast, when an inference is invalid, there is at least one valuation (i.e., a line in the truth table) where its premises are all true, and the conclusion false. Such situations are called *counter-examples*. The preceding table also gives us a counter-example for the earlier invalid consequence

from $\varphi \to \psi, \neg\varphi$ to $\neg\psi$

namely, the valuation on the third line where $\varphi \to \psi$ and $\neg\varphi$ are true but $\neg\psi$ is false.

Please note that invalidity does not say that *all* valuations making the premises true make the conclusion false. The latter would express a valid consequence again, this time, the 'refutation' of ψ (since $\neg\varphi$ is true iff φ is false):

$$\varphi_1, \ldots, \varphi_k \models \neg\psi \tag{2.23}$$

Satisfiability Finally, here is another important logical notion that gives another perspective on the same issues:

Definition 2.16 (Satisfiable) A set of formulas X (say, $\varphi_1, \ldots, \varphi_k$) is *satisfiable* if there is a valuation that makes all formulas in X true.

There is a close connection between *satisfiability* and *consistency*.

Satisfiable versus Consistent A set of formulas that does not lead to a contradiction is called a *consistent* formula set. Here 'leading to a contradiction' refers to proof rules, so this is a definition in terms of proof theory. But it is really the other side of the same coin, for a set of formulas is consistent iff the set is satisfiable. Satisfiability gives the semantic perspective on consistency.

Instead of 'not consistent' we also say *inconsistent*, which says that there is no valuation where all formulas in the set are true simultaneously.

Satisfiability (consistency) is not the same as truth: it does not say that all formulas in X are actually true, but that they could be true in some situation. This suffices for many purposes. In conversation, we often cannot check directly if what people tell us is true (think of their accounts of their holiday adventures, or the brilliance of their kids), but we often believe them as long as what they say is consistent. Also, as we noted in Chapter 1, a lawyer does not have to prove that her client is innocent, she just has to show that it is consistent with the given evidence that he is innocent.

We can test for consistency in a truth table again, looking for a line making all relevant formulas true. This is like our earlier computations, and indeed, validity and consistency are related. For instance, it follows directly from our definitions that

$$\varphi \models \psi \text{ if and only if } \{\varphi, \neg\psi\} \text{ is not consistent.} \tag{2.24}$$

Tautologies Now we look briefly at the 'laws' of our system:

Definition 2.17 (Tautology) A formula ψ that gets the value 1 in every valuation is called a *tautology*. The notation for tautologies is $\models \psi$.

Many tautologies are well-known as general laws of propositional logic. They can be used to infer quick conclusions or simplify given assertions. Here are some useful tautologies:

$$\begin{array}{ll} \text{Double Negation} & \neg\neg\varphi \leftrightarrow \varphi \\ \text{De Morgan laws} & \neg(\varphi \vee \psi) \leftrightarrow (\neg\varphi \wedge \neg\psi) \\ & \neg(\varphi \wedge \psi) \leftrightarrow (\neg\varphi \vee \neg\psi) \\ \text{Distribution laws} & (\varphi \wedge (\psi \vee \chi)) \leftrightarrow ((\varphi \wedge \psi) \vee (\varphi \wedge \chi)) \\ & (\varphi \vee (\psi \wedge \chi)) \leftrightarrow ((\varphi \vee \psi) \wedge (\varphi \vee \chi)) \end{array} \tag{2.25}$$

Check for yourself that they all get values 1 on all lines of their truth tables.

Tautologies are a special zero-premise case of valid consequences, but via a little trick, they encode all valid consequences. In fact, every valid consequence corresponds to a tautology, for it is easy to see that:

$$\varphi_1, \ldots, \varphi_k \models \psi \text{ if and only if } (\varphi_1 \wedge \ldots \wedge \varphi_k) \to \psi \text{ is a tautology} \tag{2.26}$$

Exercise 2.18 Using a truth table, determine if the two formulas

$$\neg p \to (q \vee r), \neg q$$

together logically imply

(1) $p \wedge r$.

(2) $p \vee r$.

Display the complete truth table, and use it to justify your answers to (1) and (2).

Exercise 2.19

Show using a truth table that:

- the inference from $p \to (q \wedge r)$, $\neg q$ to $\neg p$ is valid and
- the inference from $p \to (q \vee r)$, $\neg q$ to $\neg p$ is not valid.

Exercise 2.20 Check if the following are valid consequences:

(1) $\neg(q \wedge r), q \models \neg r$

(2) $\neg p \vee \neg q \vee r, q \vee r, p \models r$.

Exercise 2.21 Give truth tables for the following formulas:

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

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

(3) $(p \to (q \to r)) \to ((p \to q) \to (p \to r))$

(4) $(p \leftrightarrow (q \to r)) \leftrightarrow ((p \leftrightarrow q) \to r)$

(5) $((p \leftrightarrow q) \wedge (\neg q \to r)) \leftrightarrow (\neg(p \leftrightarrow r) \to q)$

Exercise 2.22 Which of the following pairs are *logically equivalent*? Confirm your answer using truth tables:

(1) $\varphi \to \psi$ and $\psi \to \varphi$

(2) $\varphi \to \psi$ and $\neg\psi \to \neg\varphi$

(3) $\neg(\varphi \to \psi)$ and $\varphi \vee \neg\psi$

(4) $\neg(\varphi \to \psi)$ and $\varphi \wedge \neg\psi$

(5) $\neg(\varphi \leftrightarrow \psi)$ and $\neg\varphi \leftrightarrow \neg\psi$

(6) $\neg(\varphi \leftrightarrow \psi)$ and $\neg\varphi \leftrightarrow \psi$

(7) $(\varphi \wedge \psi) \leftrightarrow (\varphi \vee \psi)$ and $\varphi \leftrightarrow \psi$

2.7 Proof

Proof: symbolic inference So far we tested inferences for validity with truth tables, staying close to the semantic meaning of the formulas. But a lot of inference happens automatically, by manipulating symbols. People usually do not reason via truth tables. They rather combine many simple proof steps that they already know, without going back to their motivation. The more such rules they have learnt, the faster their reasoning goes. Likewise, mathematicians often do formal calculation and proof via symbolic rules (think of your school algebra), and of course, computers have to do proof steps purely symbolically (as long as they have not yet learnt to think, like us, about what their actions might mean).

Logic has many formal calculi that can do proofs, and later on, we will devote a whole chapter to this topic. But in this chapter, we give you a first taste of what it means to do proof steps in a formal calculus. There is a certain pleasure and surprise to symbolic calculation that has to be experienced.

Below, we present an *axiomatic system* organized a bit like the famous geometry book of Euclid's *Elements* from Antiquity. It starts from just a few basic principles (the axioms), after which chains of many proof steps, each one simple by itself, lead to more and more, sometimes very surprising theorems.

Here is a modern axiomatic symbol game for logic:

Definition 2.23 (Axiomatization) A *proof* is a finite sequence of formulas, where each formula is either an *axiom*, or follows from previous formulas in the proof by a deduction *rule*. A formula is a *theorem* if it occurs in a proof, typically as the last formula in the sequence. A set of axioms and rules defines an *axiomatization* for a given logic.
The following is an axiomatization for propositional logic. The axioms are given in schematic form, with the formula variables that we have already seen. It means that we can put any specific formula in the place of these variables:

(1) $(\varphi \to (\psi \to \varphi))$

(2) $((\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi)))$

(3) $((\neg\varphi \to \neg\psi) \to (\psi \to \varphi))$

and there is only one deduction rule, the Modus Ponens that we have already encountered:

- if φ and $(\varphi \to \psi)$ are theorems, then ψ is also a theorem.

This axiomatization originates with the Polish logician Jan Łukasiewicz. In this system for propositional logic we may only use implication and negation symbols, and no other logical connectives, such as conjunctions. In our later section on expressivity it will be become clear why this restricted vocabulary is sufficient.

Training in axiomatic deduction will not be a key focus of this course. Still, we do want you to experience the interest of performing purely syntactic proofs, as a sort of 'symbol game' that can be interpreted later. We give one more abstract logical example here, and also one closer to practice.

Example 2.24 As an example of an axiomatic proof, we show that $p \to p$ is a theorem. This seems a self-evident tautology semantically, but now, the art is to derive it using only the rules of our game! In what follows we use well-chosen concrete instantiations of axiom schemas. For instance, the first line uses Axiom Schema 1 with the atomic proposition p for the variable formula φ and $q \to p$ for the variable formula ψ. And so on:

1. $p \to ((q \to p) \to p)$	Axiom (1)
2. $(p \to ((q \to p) \to p)) \to ((p \to (q \to p)) \to (p \to p))$	Axiom (2)
3. $(p \to (q \to p)) \to (p \to p)$	Modus Ponens, from steps $1, 2$
4. $p \to (q \to p)$	Axiom (1)
5. $p \to p$	Modus Ponens, from steps $3, 4$

It takes some skill to find such proofs by oneself. But it is actually an exciting game to many students, precisely because of the purely symbolic nature of the steps involved.

More general proofs can have certain assumptions, in addition to instances of axiom schemas. Here is an example closer to practice.

Example 2.25 Use only Modus Ponens and suitable axioms to derive the solution to the following problem. You want to throw a party, respecting people's incompatibilities. You know that:

(a) John comes if Mary or Ann comes.

(b) Ann comes if Mary does not come.

(c) If Ann comes, John does not.

Can you invite people under these constraints? There are several ways of solving this, including truth tables with update as in our next Section. But for now, can you prove what the solution must be? Here is a little help with the formal rendering:

> (i) 'If Ann comes, John does not' is the formula $a \to \neg j$, (ii) 'Ann comes if Mary does not come': $\neg m \to a$, (c) 'John comes if Mary or Ann comes': here you can rewrite to an equivalent conjunction 'John comes if Mary comes' and 'John comes if Ann comes' to produce two formulas that fall inside our language: $a \to j$, $m \to j$. Now try to give a proof just using the above axioms and rule for the solution, deriving successively that $\neg a$, m, j. Have fun!

This concludes our first glimpse of a proof game with a fixed repertoire.

System properties: soundness and completeness If all theorems of an axiomatic system are valid, the system is called *sound* , and conversely, if all valid formulas are provable theorems, the logic is called *complete*. Soundness seems an obvious requirement, as you want to rely totally on your proof procedure. The above system is sound, as you can see by noting that all axioms are tautologies, while Modus Ponens always takes tautologies to tautologies, that is, if φ and $\varphi \to \psi$ are tautologies, then ψ is also a tautology.

Completeness is a different matter, and can be harder to obtain for a given system. (Does Euclid's system of axioms suffice for proving all truths of geometry? The answer took centuries of investigation and reformulation of the system.) The above proof system is indeed complete, and so are the proof systems that we will present in later chapters. But showing that completeness holds can be hard. The completeness of predicate logic, that we will discuss in later chapters, was one of the first deep results in modern logic, discovered by the then 23-year old Kurt Gödel in his 1929 dissertation.

Axiomatic deduction is only one of many proof methods used in logic. Others include *natural deduction* (used a lot in logical Proof Theory) and *resolution* (used in many automated theorem provers). Chapter 9 in Part III of the book will take you much further into this area.

2.8 Information Update

With all this in place, we can now also define our earlier notions of *information* structure and information growth:

> The information content of a formula φ is the set $\mathsf{MOD}(\varphi)$ of its *models*, that is, the valuations that assign the formula φ the truth-value 1.

You can think of this as the range of possible situations that φ leaves open. Note that the more possibilities are left open by a formula φ, the less information φ contains. Formulas that leave many possibilities open correspond to information states with much uncertainty. Formulas that leave just one possibility open — that have just one satisfying valuation — leave no uncertainty at all about what the situation is like.

Information update by elimination of possibilities Here is the dynamics that changes such information states:

> An update with new information ψ reduces the current set of models X to the overlap or *intersection* of X and $\mathsf{MOD}(\psi)$. The valuations in X that assign the value 0 to ψ are *eliminated*.

Thus, propositional logic gives an account of basic cognitive dynamics, where information states (sets of satisfying valuations) shrink as new information comes in: growth of knowledge is loss of uncertainty.

We have seen earlier how this worked with simple inferences like

> ‘from $p \vee q, \neg p$ to q’,

if we assume that the premises update an initial information state of no information (maximal uncertainty: all valuations still present).

As a second example, we return to an earlier question in Section 2.3 (see Exercise 2.3)

> What information is given by $p \vee q, \neg p \vee r$?

Here are the update stages:

$$\begin{array}{ll} \text{initial state} & \{pqr, pq\overline{r}, p\overline{q}r, p\overline{q}\overline{r}, \overline{p}qr, \overline{p}q\overline{r}, \overline{p}\overline{q}r, \overline{p}\overline{q}\overline{r}\} \\ \text{update with } p \vee q & \{pqr, pq\overline{r}, p\overline{q}r, p\overline{q}\overline{r}, \overline{p}qr, \overline{p}q\overline{r}\} \\ \text{update with } \neg p \vee r & \{pqr, p\overline{q}r, \overline{p}qr, \overline{p}q\overline{r}\} \end{array} \quad (2.27)$$

We can conclude whatever is true in all of the remaining four states. One valid conclusion is the inclusive disjunction $q \vee r$, and this is indeed the one used in the so-called *resolution rule* of many automated reasoning systems. But actually, the two given premises are stronger than the inference $q \vee r$. The situation $pq\overline{r}$ is not among the ones that are left after the updates in (2.27), but $q \vee r$ is obviously true in this situation. One trivial way of really getting all content of the premises is of course just their conjunction:

$$(p \vee q) \wedge (\neg p \vee r). \quad (2.28)$$

But there is alo a disjunctive form that precisely describes the final information set $\{pqr, p\overline{q}r, \overline{p}qr, \overline{p}q\overline{r}\}$:

$$(p \wedge r) \vee (\neg p \wedge q). \quad (2.29)$$

In practice, we want convenient descriptions of information states, and later on we will look at some principles of Boolean Algebra that can help us with this.

Planning Other information scenarios arise in planning problems. Recall the Party Problem in Section 2.7 (Example 2.25). Can you invite people under the three constraints? One sure way is computing information updates from an initial state of no information about constraints:

$$\{maj, ma\overline{j}, m\overline{a}j, m\overline{a}\overline{j}, \overline{m}aj, \overline{m}a\overline{j}, \overline{m}\overline{a}j, \overline{m}\overline{a}\overline{j}\} \quad (2.30)$$

Now the three given premises update this initial information state, by removing options incompatible with them. In successive steps, (a), (b), (c) give the following reductions:

$$\begin{array}{lll} \text{(a)} & (m \vee a) \to j & \{maj, m\bar{a}j, \overline{m}aj, \overline{m}\overline{a}j, \overline{ma}\overline{j}\} \\ \text{(b)} & \neg m \to a & \{ma\jmath, m\bar{a}j, \overline{m}aj\} \\ \text{(c)} & a \to \neg j & \{m\bar{a}j\} \end{array} \tag{2.31}$$

Incidentally, this is a unique solution for the stated constraints – but this need not at all be the case in general: there could be none, or more than one option remaining, too.

Games as information processing Our update process describes the information flow in games like *Master Mind*, where players have to guess the correct position of some hidden coloured pegs. In each round, she can make a guess, that gets evaluated by black marks for colours in correct positions, and white marks for colours that do occur, but placed in wrong positions.

For instance, let there be four possible colours red, white, blue, orange, and three positions, with a hidden correct sequence red-white-blue. Here is a table for a possible run of the game, indicating the information game in successive answers:

guess	answer	possibilities remaining
START		24
red, orange, white	•◦	6
white, orange, blue	•◦	2
blue, orange, red	◦◦	1

You will find it useful to do the updates, and see why the given numbers are correct.

Master Mind is not really interactive (a machine could provide the answers to your guesses), though new interactive variants are used these days in psychological experiments about children's reasoning with different agents. Information update with different agents, as well as more realistic games will be studied in Chapters 5 and 7. Elimination of possibilities is still fundamental there, so what you learnt here has a broad thrust.

2.9 Expressiveness

A logical language is not just an auxiliary tool for studying inferences and updates. It is also a language that can be used for the common things we have languages for: stating truths (and lies) about situations, communicating important facts to others, and so on. In this light, a very fundamental issue about a logical language is its *expressiveness*. What can we say with it?

For a start, propositional logic may look very poor. We can combine sentences, but we cannot look 'inside' them: "Horatio Nelson died at Trafalgar" is just an atomic proposition, say p. But the real issue how well it does within its own compass. And then we find a pleasant surprise:

Propositional logic is quite expressive! In total, there are sixteen possible Boolean operations (truth value assignments) with two arguments: count options in the truth table. This is many more than the number of binary operators we have in our language. Some of these correspond to serious expressions in natural language. In particular, the *exclusive disjunction* $\varphi \oplus \psi$ corresponds to the natural language phrasing 'either-φ-or-ψ'. It has the following truth table, compare it to the one for disjunction $\vee$:

φ	ψ	$\varphi \oplus \psi$	$\varphi \vee \psi$
0	0	**0**	**0**
0	1	**1**	**1**
1	0	**1**	**1**
1	1	**0**	**1**

Now note that we could get the same truth table by *defining* exclusive disjunction $\varphi \oplus \psi$ in terms of notions that we already had:

$$(\varphi \vee \psi) \wedge \neg(\varphi \wedge \psi) \text{ or, alternatively } \neg(\varphi \leftrightarrow \psi) \tag{2.32}$$

More generally, it is not hard to prove that

> All sixteen possible binary propositional operations are *definable* in terms of just the three operations $\neg$, $\wedge$ and $\vee$.

As an illustration,

> the implication $\varphi \to \psi$ has the same truth table as $\neg\varphi \vee \psi$ and as $\neg(\varphi \wedge \neg\psi)$.

In fact, even $\neg, \wedge$ alone suffice for defining all possible operations, and also $\neg, \vee$ alone, and $\neg, \to$ alone. As you will recall, the latter fact was used in the axiomatization of propositional logic in the section on proof.

Exercise 2.26 Define all connectives in terms of $\neg$ and $\wedge$.

Exercise 2.27 Define all connectives in terms of $\neg$ and $\to$.

Indeed, there is even an operation that can define all propositional operations by itself, the *Sheffer stroke*

$$\varphi \mid \psi,$$

defined as $\neg\varphi \vee \neg\psi$.

Now you know how expressive our language is on its own turf: it can express anything we want to say about combination of two-valued propositions.

This is just one of many interesting features of definability in propositional logic. Here is another, that we state without giving details. Every propositional logical formula, no matter how complex, is equivalent to a conjunction of disjunctions of proposition letters or their negations. This is called the 'conjunctive normal form' (there is also a disjunctive normal form). For instance, the conjunctive normal form for the earlier exclusive disjunction is

$$(\varphi \vee \psi) \wedge (\neg\varphi \vee \neg\psi). \tag{2.33}$$

Seeking a balance Clearly, there are many things that we cannot express in propositional logic. The following chapters are about more expressive languages, such as predicate logic or epistemic logic. Even so, an important thing to keep in mind is a *Balance*. In logic as in science, the art is to stay as simple as possible: 'Small is Beautiful'. A poor language may have special properties that make it elegant or useful. Propositional logic is very successful in bringing out basic reasoning patterns, and moreover, its very poverty leads to elegant and simple semantics and proof methods. In richer systems, the latter become more baroque, and sometimes essentially more complex.

This completes the standard part of this chapter. Next comes a sequence of special topics that will help you see where propositional logic lives in a larger scientific world.

2.10 Outlook — Logic, Mathematics, Computation

Studying logical phenomena via mathematical systems has proved a powerful method historically. Thinking about our language of logical forms yields general insights into expressive power, as we have just learnt. But also, thinking about a system of all validities per se yields new insights that can be used in many settings. Here are some examples:

Boolean algebra The system of laws shows many beautiful regularities. For instance, De Morgan and Distribution laws came in pairs with conjunction and disjunction interchanged. This 'duality' is general, and it reflects the close analogy between propositional logic and binary arithmetic (arithmetic with just 0 and 1, where every number is represented in the binary, or base-2, number system). In particular, the truth tables are just laws of binary arithmetic when we read:

> $\vee$ as the maximum of two numbers, $\wedge$ as the minimum,
> and $\neg$ as flipping 0 and 1.

Suppressing details, distribution for conjunction over disjunction then matches the arithmetical distribution law $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$. But binary arithmetic is even better-behaved: it also validates another distribution law $x + (y \cdot z) = (x + y) \cdot (x + z)$ that does not hold for numbers in general (try some numbers, and you will see). We will pursue such connections between logic and computation in more detail in later chapters.

Abstraction and application Logical systems originally arose out of concrete practice. But conversely, once we have such abstract systems, new concrete interpretations may be found. Boolean algebra is an example. It describes a whole range of phenomena: propositional reasoning, binary arithmetic, reasoning with sets (where $\neg$ is complement, $\wedge$ intersection, and $\vee$ union), and even electrical switching circuits where conjunction is serial composition of networks, and disjunction is *parallel composition*. Thus, one and the same formula says lots of things!

For instance, consider one single abstract principle, the Boolean law of 'Absorption':

$$\varphi \leftrightarrow (\varphi \wedge (\varphi \vee \psi)) \tag{2.34}$$

This is a tautology for propositional reasoning that helps remove redundancies from discourse (or when used in the opposite direction, helping you make simple things sound complicated). Next, in binary arithmetic, it expresses a valid equation

$$x = x \text{ min } (x \text{ max } y) \tag{2.35}$$

about computing with minima and maxima. In set theory, Absorption is the valid principle

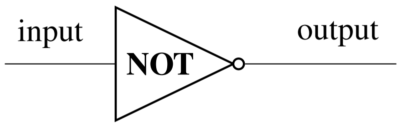
$$X = X \cap (X \cup Y) \tag{2.36}$$

which says that the intersection ('overlap') of the set X and the union of X and Y is the same as the set X itself.

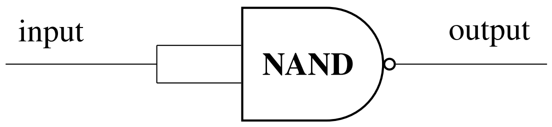
In a similar way, propositional logic plays a role in the design of logical (electronic) circuits. A NAND gate is an electronic circuit that behaves like the Sheffer stroke. The gate has two inputs and an output. If both inputs are 1 (carry a high voltage) then the output is low (carries a low voltage). For all other input combinations (high and low, low and high, low and low) the output is high. Since any propositional connective can be defined with just the Sheffer stroke, any desirable logical circuit can be built from a combination of NAND gates. Here is how negation is defined with the Sheffer stroke:

$$\varphi \mid \varphi.$$

The same principle can be used to build a NOT gate from a NAND gate:



NOT gate



NOT gate implemented with NAND gate

Thus we see a glimpse of a general reality: boolean algebra underlies real Boolean circuits in computers. The details of how this works can be found in many sources, including many sites on the internet.

Soundness and completeness The same general properties that we stated for our proof system also make sense here. In the nineteenth century, George Boole gave a complete algebraic analysis of propositional logic for reasoning with sentential operators like ‘not’, ‘and’, ‘or’, that has become famous as the ‘Boolean algebra’ that underlies the switching circuits of your computer. Here is what such a system looks like, with variables for propositions that can be true (1) or false (0), and operations $-$ for ‘not’, $\cdot$ for ‘and’, and $+$ for ‘or’ in the sense of binary arithmetic:

$$\begin{array}{rclrcl}
x+(y+z) & = & (x+y)+z & x\cdot(y\cdot z) & = & (x\cdot y)\cdot z \\
x+y & = & y+x & x\cdot y & = & y\cdot x \\
x+x & = & x & x\cdot x & = & x \\
x+(y\cdot z) & = & (x+y)\cdot(x+z) & x\cdot(y+z) & = & (x\cdot y)+(x\cdot z) \\
x+(x\cdot y) & = & x & x\cdot(x+y) & = & x \\
-(x+y) & = & -x\cdot -y & -(x\cdot y) & = & -x+-y \\
x+0 & = & x & x\cdot 0 & = & 0 \\
x+1 & = & 1 & x\cdot 1 & = & x \\
x+-x & = & 1 & x\cdot -x & = & 0 \\
\end{array}$$
$$--x = x \tag{2.37}$$

It is easy to see that these equations correspond to valid tautologies, when read as equivalences between propositional formulas. Thus we have soundness: the calculus proves only valid principles. Conversely, Boolean algebra is also complete, and any valid equation can be derived from it by ordinary algebraic manipulations.

Computational complexity Propositional logic is tied up with computation in many ways, as we have seen in this chapter. In particular, truth tables make testing for logical validity a simple procedure that can be done mechanically. And indeed, there exist computer programs for all of the tasks in this chapter. Within the compass of our simple language, this realizes a famous historical project: Leibniz's 'Calculus Ratiocinator' around 1700, which proposed that all reasoning can be reduced to computation. Indeed, there is a long history of 'logical machines' for carrying out inference tasks, going back to the Middle Ages.

Still, all this is computability in principle, and things are delicate in practice. A mechanical method with simple steps can still be highly complex when very many of these steps must be made. Consider truth tables. Computing truth values on a single line for a given formula goes fast. Earlier on we wrote successive truth values in the construction tree, and the number of time steps required for this is 'linear' (if the line has twice as many symbols, the computation takes roughly twice as many steps):

Computing a truth value for a formula takes linear time,

of the same order as the number of symbols in the formula. But now consider the whole truth table for a formula. With n atomic propositions we need 2^n lines, leading to *exponential growth* for the table in the size of the input:

Computing a truth table for validity takes exponential time.

This quickly outgrows the powers of even the fastest current computers. Therefore, smarter methods have been investigated, cutting down on the number of steps needed to test for validity — such as the semantic tableaus that you will see in Chapter 7. But it was always found that, in the worst case with difficult input formulas, these still require exponential time.

This is no coincidence. The exact computational complexity of validity in propositional logic is unknown: there may still be a faster method than existing ones that would work with a polynomial bound on processing time, though most experts doubt this. Determining this exact complexity is the essence of the famous

'$\mathbf{P} = \mathbf{NP}$ Problem' ,

that occurs on the famous 2000 Millennium List of open problems in mathematics posed by the Clay Mathematics Institute.

This problem is urgent since it has been shown that many basic computational tasks reduce to solving problems of validity and consistency in propositional logic. Thus, on its two-thousandth anniversary, propositional logic still poses deep problems.

Higher expressive power and undecidability Whether highly complex or not, the problem of testing for validity in propositional logic is *decidable*: there exists a mechanical method that computes the answer, at least in principle. Thus it may seem that computers can always do the job of logicians. But things change when we move to logics with higher expressive power, such as the predicate logic of Chapter 4 with quantifiers 'all', and 'some'. It is known from the work of Gödel, Turing, and others in the 1930s that there is no mechanical method at all for testing validity of predicate-logical inferences: these major systems pay a price for their greater expressive power: they are *undecidable*.

2.11 Outlook — Logic and Practice

The art of modelling To apply an abstract system like propositional logic, you need 'modelling skills'. For instance, we have already observed that you need to translate from natural language sentences to logical forms to get at the essence of an inference. This often takes practice, but it can be fun, witness the popular logic puzzles in commercial journals. Here is one simple example.

Propositional logic has generated many puzzles. The next exercise is from Raymond Smullyan's *The Lady or the Tiger?*, Penguin Books, 1982.

Exercise 2.28 Consider these two room signs:

- A – In this room there is a lady, and in the other one there is a tiger.
- B – In one of these rooms, there is a lady, and in one of them there is a tiger"

One of these signs is true, the other false. Behind which door is the lady?

But beyond this recreational aspect, propositional logic also applies to more serious areas of reasoning: witness, e.g., a whole literature on using propositional logic in legal reasoning. More technically, propositional logic has been applied to a wide variety of computational tasks, from Boolean circuits in your computer to complex train movements at the shunting yards of the Dutch Railways.

Such applications are not routine, and require creative skills.

Improving practice Training in propositional logic is also used to improve practical skills. This is an old tradition. Legend has it that medieval logic exams checked students' real-time skills as follows:

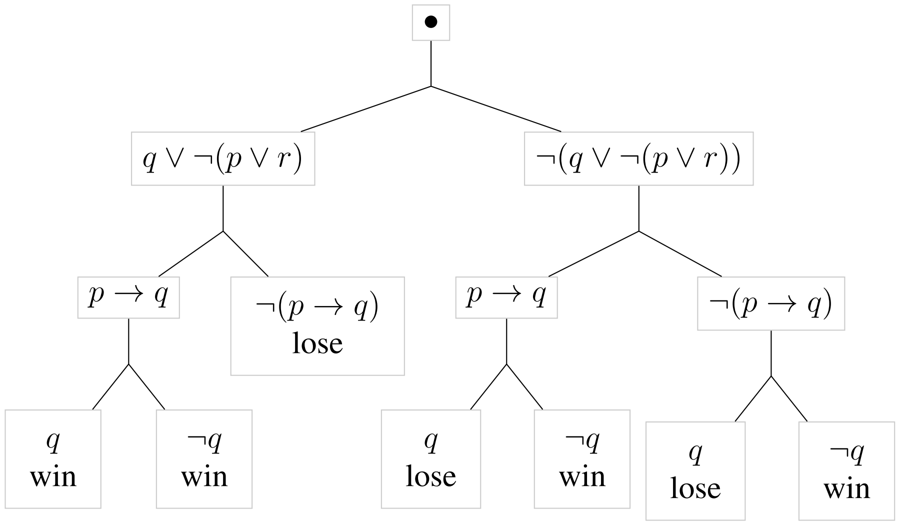
> *Obligatio Game* A finite number of rounds is chosen, the severity of the exam. The teacher gives the student successive assertions $\varphi_1, \ldots, \varphi_n$ that she has to 'accept' or 'reject' as they are put forward. In the former case, φ_i is added to

> the students stock of commitments — in the latter, the negation $\neg\varphi_i$ is added. The student passes if she maintains consistency throughout.

Suppose that a student is exposed to the following three statements:

$$(1)\ \ q \vee \neg(p \vee r), (2)\ \ p \rightarrow q, (3)\ \ q. \tag{2.38}$$

Here is one possible run. If you say YES to (1), you must say YES to (2), since it follows but then you can say either YES or NO to (3), since it is independent. Next, if you say NO to (1), you can say either YES or NO to (2), but then, in both cases, you must say NO to (3), as it follows from the negation of (1). The whole picture is:



This may be viewed as a *game tree* with all possible plays including the winning branches. (A complete tree would include Teacher's choices of the next assertion from some given set — possibly influenced by what Student has answered so far.) Either way, the tree will show that, as is only fair on exams, the student has a winning strategy for this game of consistency management. The logical reason is this:

> Any consistent set of assertions can always be consistently expanded with at least one of the propositions φ, $\neg\varphi$.

The winning strategy based on this seems to require consistency checking at each stage, a hard computational problem. A simpler strategy for the student is this:

> choose one model beforehand (say, a valuation making each atom true), and evaluate each incoming assertion there, giving the obvious answers.

2.12 Outlook — Logic and Cognition

But how do logical systems relate to our daily practice where we reason and try to make sense without consciously thinking about how we do it? One interface with reality has occurred a number of times now:

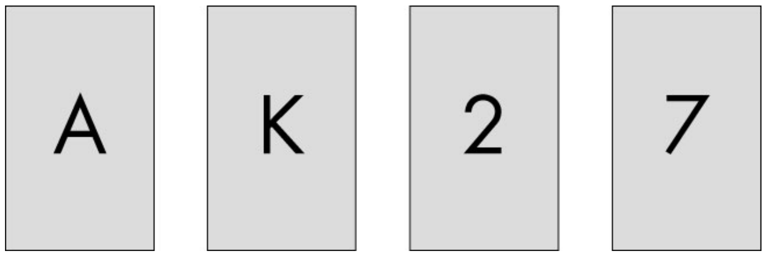
Logic and linguistics Natural languages have a much richer repertoire of meanings than the formal language of propositional logic. For instance, the expression *and* also has frequent non-Boolean readings. "John and Mary quarrelled" does not mean that "John quarrelled and Mary quarrelled", Likewise, we already noted that conditional expressions like *if ... then* do not behave exactly like the truth-table conditional. In particular, a false antecedent does not necessarily make them true: "If I were rich, I would be generous" does not follow from my not being rich.

But all this does not mean that logical methods do not apply. In fact, the divergence has turned a creative advantage. The richer structure of natural language has been an inexhaustible source of new logical theory. For instance, propositional logic has been generalized to work with more than two truth values to model vague or indeterminate uses of language, and the study of various sorts of conditional expressions has become a flourishing subdiscipline where logicians and linguists work together.

Logic and cognitive psychology The relation between logic and psychology has been somewhat stormier. It has been claimed by psychologists that everyday reasoning is highly non-logical. Here is a famous example.

The Wason selection task is a logic puzzle that states the following question:

> You are shown a set of four cards placed on a table each of which has a number on one side and a colored patch on the other side. The visible faces of the cards show 2, 7, A and K. Which card(s) should you turn over in order to test the truth of the proposition that if a card shows an even number on one face, then its opposite face shows a vowel?



The Wason selection task

Here is the correct response according to the logic of this chapter:

> turn the cards showing 2 and K, but no other card.

The reason is this: to test the implication EVEN $\rightarrow$ VOWEL, we clearly need to check the card with the even number 2, but also should not forget the refutation case discussed several times before: if the card does not show a vowel, we need to make sure that it did not have an even number. Now the results of the experiment, repeated over many decades:

> most people either (a) turn the 2 only, or (b) they turn the 2 and the A card.

Psychologists have suggested many explanations, including a 'confirmation bias' (refutation comes less natural to us) and an 'association bias' (red is mentioned so we check it). This seems to suggest that real reasoning is very different from what logic says.

However, the selection task tends to produce the correct logical response when presented in more concrete contexts that the experimental subjects are familiar with. For example, if the rule is 'If you are drinking alcohol, then you must be over 18', and the cards have an age on one side and a beverage on the other, e.g., '17', 'beer', '22', 'coke', most people have no difficulty in selecting the correct cards ('17' and 'beer'). Psychologists have used this as another argument against logic: the two settings have the same logical form, but very different behaviour results from familiarity effects.

More information on this famous experiment can be found on the webpage `http://en.wikipedia.org/wiki/Wason_selection_task`.

Frankly, all this polemics is not interesting. Clearly, people are not irrational, and if they ignored logic all the time, extracting the wrong information from the data at their disposal, it is hard to see how our species could survive. What seems to be the case is rather an issue of *representation* of reasoning tasks, and additional principles that play a role there. Moreover, the variation in outcomes fits with a conspicuous trend in modern logic, namely, the study of *task-dependent* forms of inference, whose rules may differ from the strict standards set in this chapter. These include more heuristic 'default rules' that are not valid in our strict sense, but that can be used until some problem arises that requires a revision of what we concluded so far.

But let us give the last word to George Boole, often considered the father of the purely mathematical approach to (propositional) logic. The title of his great work "The Laws of Thought" would seem to sit uneasily with a diehard normative mathematical perspective. But toward the end of the book, Boole remarks that he is serious about the title: the laws of propositional logic describe essential human thought. He also acknowledges that human reasoning often deviates from this canon. What that means is, he says, that there are *further laws* of human thought that still need to be discovered. That is what the modern interface of logic and cognitive science is about.

Further Exercises

Exercise 2.29 Prove that all propositional connectives are definable with the 'Sheffer stroke'

$$\varphi \mid \psi,$$

defined by $\neg\varphi \vee \neg\psi$.

Exercise 2.30 In how many ways can you win the following *obligatio* game?

$$(1)\ \ (p \to q) \vee (r \to q), (2)\ \ \neg((p \wedge r) \to q), (3)\ \ q.$$

Exercise 2.31 Consider the following formula:

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

The logical symbols in this formula are all the symbols except parentheses and propositional variables. As you can see, the formula has 11 logical symbols. Answer the following questions:

(1) How many truth value entries does the truth table for this formula have. How does that number depend on the number of logical symbols?

(2) The truth table for a formula with 3 propositional variables has $2^3 = 8$ rows. How many entries in the truth table for such a formula do you have to compute (in the worst case) in order to find out if the formula is valid or not, given that you know that the formula has n logical symbols?

Summary *You have now seen your first logical system, and know how to reason in an exact mathematical manner with propositions. In particular, you have learnt these skills:*

- *read and write propositional formulas,*
- *translate simple natural language sentences into formulas,*
- *compute truth tables for various purposes,*
- *test validity of inferences,*
- *compute updates of information states,*
- *do some very simple formal proofs.*

In addition, you now have a working knowledge of the notions of

> *syntax, semantics, valuation, truth, valid consequence, tautology, consistency, axiomatic proof, expressive power, logical system.*

Finally, you have seen a first glimpse of connections between propositional logic and mathematical proof, computation, complexity, and some cognitive topics, namely, natural language and psychological experiments.

Further Reading Propositional logic was already known to the Stoic philosophers. See [Mat73] for an account. Propositional logic is fully developed in the famous book of George Boole [Boo54]. Boole gives an algebraic treatment of the logic of propositions. This kind of algebra is now known as Boolean algebra. A modern treatment of Boolean algebra is given in [GH09]. If you are in for some logic entertainment you should consult [CSI08] or the famous logic puzzle books by Raymond Smullyan [Smu09, Smu11].