Part I

Propositional Logic

1 Classical Logic and the Material Conditional

1.1 Introduction

1.1.1 The first purpose of this chapter is to review classical propositional logic, including semantic tableaux. The chapter also sets out some basic terminology and notational conventions for the rest of the book.

1.1.2 In the second half of the chapter we also look at the notion of the conditional that classical propositional logic gives, and, specifically, at some of its shortcomings.

1.1.3 The point of logic is to give an account of the notion of validity: what follows from what. Standardly, validity is defined for inferences couched in a formal language, a language with a well-defined vocabulary and grammar, the *object language*. The relationship of the symbols of the formal language to the words of the vernacular, English in this case, is always an important issue.

1.1.4 Accounts of validity themselves are in a language that is normally distinct from the object language. This is called the *metalanguage*. In our case, this is simply mathematical English. Note that 'iff' means 'if and only if'.

1.1.5 It is also standard to define two notions of validity. The first is *semantic*. A valid inference is one that *preserves truth*, in a certain sense. Specifically, every interpretation (that is, crudely, a way of assigning truth values) that makes all the premises true makes the conclusion true. We use the metalinguistic symbol '$\models$' for this. What distinguishes different logics is the different notions of interpretation they employ.

1.1.6 The second notion of validity is *proof-theoretic*. Validity is defined in terms of some purely formal procedure (that is, one that makes reference only to the symbols of the inference). We use the metalinguistic symbol '$\vdash$' for this notion of validity. In our case, this procedure will (mainly) be one employing tableaux. What distinguish different logics here are the different tableau procedures employed.

1.1.7 Most contemporary logicians would take the semantic notion of validity to be more fundamental than the proof-theoretic one, though the matter is certainly debatable. However, given a semantic notion of validity, it is always useful to have a proof-theoretic notion that corresponds to it, in the sense that the two definitions always give the same answers. If every proof-theoretically valid inference is semantically valid (so that $\vdash$ entails $\models$) the proof-theory is said to be *sound*. If every semantically valid inference is proof-theoretically valid (so that $\models$ entails $\vdash$) the proof-theory is said to be *complete*.

1.2 The Syntax of the Object Language

1.2.1 The symbols of the object language of the propositional calculus are an infinite number of propositional parameters:[1] $p_0, p_1, p_2, \ldots$; the connectives: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\supset$ (material conditional), $\equiv$ (material equivalence); and the punctuation marks: (,).

1.2.2 The (well-formed) formulas of the language comprise all, and only, the strings of symbols that can be generated recursively from the propositional parameters by the following rule:

If A and B are formulas, so are $\neg A$, $(A \vee B)$, $(A \wedge B)$, $(A \supset B)$, $(A \equiv B)$.

1.2.3 I will explain a number of important notational conventions here. I use capital Roman letters, $A, B, C, \ldots$, to represent arbitrary formulas of the object language. Lower-case Roman letters, p, q, r, $\ldots$, represent arbitrary,

[1] These are often called 'propositional variables'.

but distinct, propositional parameters. I will always omit outermost parentheses of formulas if there are any. So, for example, I write $(A \supset (B \vee \neg C))$ simply as $A \supset (B \vee \neg C)$. Upper-case Greek letters, $\Sigma, \Pi, \ldots$, represent arbitrary sets of formulas; the empty set, however, is denoted by the (lower case) ϕ, in the standard way. I often write a finite set, $\{A_1, A_2, \ldots, A_n\}$, simply as $A_1, A_2, \ldots, A_n$.

1.3 Semantic Validity

1.3.1 An *interpretation* of the language is a function, ν, which assigns to each propositional parameter either 1 (true), or 0 (false). Thus, we write things such as $\nu(p) = 1$ and $\nu(q) = 0$.

1.3.2 Given an interpretation of the language, ν, this is extended to a function that assigns every formula a truth value, by the following recursive clauses, which mirror the syntactic recursive clauses:[2]

$\nu(\neg A) = 1$ if $\nu(A) = 0$, and 0 otherwise.
$\nu(A \wedge B) = 1$ if $\nu(A) = \nu(B) = 1$, and 0 otherwise.
$\nu(A \vee B) = 1$ if $\nu(A) = 1$ or $\nu(B) = 1$, and 0 otherwise.
$\nu(A \supset B) = 1$ if $\nu(A) = 0$ or $\nu(B) = 1$, and 0 otherwise.
$\nu(A \equiv B) = 1$ if $\nu(A) = \nu(B)$, and 0 otherwise.

1.3.3 Let Σ be any set of formulas (the premises); then A (the conclusion) is a *semantic consequence* of Σ ($\Sigma \models A$) iff there is no interpretation that makes all the members of Σ true and A false, that is, every interpretation that makes all the members of Σ true makes A true. '$\Sigma \not\models A$' means that it is not the case that $\Sigma \models A$.

1.3.4 A is a *logical truth* (*tautology*) ($\models A$) iff it is a semantic consequence of the empty set of premises ($\phi \models A$), that is, every interpretation makes A true.

[2] The reader might be more familiar with the information contained in these clauses when it is depicted in the form of a table, usually called a *truth table*, such as the one for *conjunction* displayed:

$\wedge$	**1**	**0**
1	1	0
0	0	0

1.4 Tableaux

1.4.1 A *tree* is a structure that looks, generally, like this:[3]

$$\begin{tikzcd}
& \cdot \arrow[d] & & \\
& \cdot \arrow[dl] \arrow[dr] & & \\
\cdot \arrow[d] & & \cdot \arrow[dl] \arrow[dr] & \\
\cdot & \cdot & & \cdot
\end{tikzcd}$$

The dots are called *nodes*. The node at the top is called the *root*. The nodes at the bottom are called *tips*. Any path from the root down a series of arrows as far as you can go is called a *branch*. (Later on we will have trees with infinite branches, but not yet.)

1.4.2 To test an inference for validity, we construct a tableau which begins with a single branch at whose nodes occur the premises (if there are any) and the negation of the conclusion. We will call this the *initial list*. We then apply rules which allow us to extend this branch. The rules for the conditional are as follows:

$$\begin{tikzcd}
& A \supset B \arrow[dl] \arrow[dr] & & \neg(A \supset B) \arrow[d] \\
\neg A & & B & A \arrow[d] \\
& & & \neg B
\end{tikzcd}$$

The rule on the right is to be interpreted as follows. If we have a formula $\neg(A \supset B)$ at a node, then every branch that goes through that node is extended with two further nodes, one for A and one for $\neg B$. The rule on the left is interpreted similarly: if we have a formula $A \supset B$ at a node, then every branch that goes through that node is split at its tip into two branches; one contains a node for $\neg A$; the other contains a node for B.

[3] Strictly speaking, for those who want the precise mathematical definition, it is a partial order with a unique maximum element, x_0, such that for any element, x_n, there is a unique finite chain of elements $x_n \leq x_{n-1} \leq \cdots \leq x_1 \leq x_0$.

1.4.3 For example, to test the inference whose premises are $A \supset B$, $B \supset C$, and whose conclusion is $A \supset C$, we construct the following tree:

$$\begin{tikzcd}
 & & A \supset B \arrow[d] & & \\
 & & B \supset C \arrow[d] & & \\
 & & \neg(A \supset C) \arrow[d] & & \\
 & & A \arrow[d] & & \\
 & & \neg C \arrow[dl] \arrow[dr] & & \\
 & \neg A \arrow[dl] \arrow[d] & & B \arrow[d] \arrow[dr] & \\
\neg B & C & & \neg B & C \\
\times & \times & & \times & \times
\end{tikzcd}$$

The first three formulas are the premises and negated conclusion. The next two formulas are produced by the rule for the negated conditional applied to the negated conclusion; the first split on the branch is produced by applying the rule for the conditional to the first premise; the next splits are produced by applying the same rule to the second premise. (Ignore the '$\times$'s: we will come back to those in a moment.)

1.4.4 The other connectives also have rules, which are as follows.

$$\begin{tikzcd}
\neg\neg A \arrow[d] \\
A
\end{tikzcd}$$

$$\begin{tikzcd}
 & A \vee B \arrow[dl] \arrow[dr] & & \neg(A \vee B) \arrow[d] \\
A & & B & \neg A \arrow[d] \\
 & & & \neg B
\end{tikzcd}$$

$$\begin{tikzcd}
 & \neg(A \wedge B) \arrow[dl] \arrow[dr] & & A \wedge B \arrow[d] \\
\neg A & & \neg B & A \arrow[d] \\
 & & & B
\end{tikzcd}$$

$$\begin{tikzcd}
 & A \equiv B \arrow[dl] \arrow[dr] & & & \neg(A \equiv B) \arrow[dl] \arrow[dr] & \\
A \arrow[d] & & \neg A \arrow[d] & A \arrow[d] & & \neg A \arrow[d] \\
B & & \neg B & \neg B & & B
\end{tikzcd}$$

Intuitively, what a tableau means is the following. If we apply a rule to a formula, then if that formula is true in an interpretation, so are the formulas below on at least one of the branches that the rule generates. (Of course, there may be only one such branch.) This is a useful mnemonic for remembering the rules. It must be stressed, though, that officially the rules are purely formal.

1.4.5 A tableau is *complete* iff every rule that can be applied has been applied. By applying the rules over and over, we may always construct a complete tableau. In the present case, the branches of a completed tableau are always finite,[4] but in the tableaux of some subsequent chapters they may be infinite.

1.4.6 A branch is *closed* iff there are formulas of the form A and $\neg A$ on two of its nodes; otherwise it is *open*. A closed branch is indicated by writing an $\times$ at the bottom. A *tableau* itself is closed iff every branch is closed; otherwise it is open. Thus the tableau of 1.4.3 is closed: the leftmost branch contains A and $\neg A$; the next contains A and $\neg A$ (and C and $\neg C$); the next contains B and $\neg B$; the rightmost contains C and $\neg C$.

1.4.7 A is a proof-theoretic consequence of the set of formulas Σ ($\Sigma \vdash A$) iff there is a complete tree whose initial list comprises the members of Σ and the negation of A, and which is closed. We write $\vdash A$ to mean that $\phi \vdash A$,

[4] This is not entirely obvious, though it is not difficult to prove.

that is, where the initial list of the tableau comprises just $\neg A$. '$\Sigma \nvdash A$' means that it is not the case that $\Sigma \vdash A$.[5]

1.4.8 Thus, the tree of 1.4.3 shows that $A \supset B, B \supset C \vdash A \supset C$. Here is another, to show that $\vdash ((A \supset B) \wedge (A \supset C)) \supset (A \supset (B \wedge C))$. To save space, we omit arrows where a branch does not divide.

$$\begin{tikzcd}
 & & \neg(((A \supset B) \wedge (A \supset C)) \supset (A \supset (B \wedge C))) & & & \\
 & & (A \supset B) \wedge (A \supset C) & & & \\
 & & \neg(A \supset (B \wedge C)) & & & \\
 & & (A \supset B) & & & \\
 & & (A \supset C) & & & \\
 & & A & & & \\
 & & \neg(B \wedge C) \arrow[dl] \arrow[dr] & & & \\
 & \neg B \arrow[dl] \arrow[d] & & \neg C \arrow[d] \arrow[dr] & & \\
\neg A & B & & \neg A & B \arrow[d] \arrow[dr] & \\
\times & \times & & \times & \neg A & C \\
 & & & & \times & \times
\end{tikzcd}$$

Note that when we find a contradiction on a branch, there is no point in continuing it further. We know that the branch is going to close, whatever else is added to it. Hence, we need not bother to extend a branch as soon as it is found to close. Notice also that, wherever possible, we apply rules that do not split branches before rules that split branches. Though this is not essential, it keeps the tableau simpler, and is therefore useful practically.

1.4.9 In practice, it is also a useful idea to put a tick at the side of a formula once one has applied a rule to it. Then one knows that one can forget about it.

[5] There may, in fact, be several completed trees for an inference, depending upon the order of the premises in the initial list and the order in which rules are applied. Fortunately, they all give the same result, though this is not entirely obvious. See 1.14, problem 5.

1.5 Counter-models

1.5.1 Here is another example, to show that $(p \supset q) \vee (r \supset q) \nvdash (p \vee r) \supset q$.

$$\begin{tikzcd}
 & & & \mathbf{(p \supset q) \vee (r \supset q)} & & & \\
 & & & \neg\mathbf{((p \vee r) \supset q)} & & & \\
 & & & \mathbf{(p \vee r)} & & & \\
 & & & \neg\mathbf{q} \arrow[dl] \arrow[dr] & & & \\
 & & \mathbf{(p \supset q)} \arrow[dl] \arrow[d] & & (r \supset q) \arrow[d] \arrow[dr] & & \\
 & \neg\mathbf{p} \arrow[ddl] \arrow[dd] & q & & \neg r \arrow[dd] \arrow[ddr] & q & \\
 & & \times & & & & \times \\
p & \mathbf{r} & & & p & r & \\
\times & & & & & \times &
\end{tikzcd}$$

The tableau has two open branches. The leftmost one is emphasised in bold for future reference.

1.5.2 The tableau procedure is, in effect, a systematic search for an interpretation that makes all the formulas on the initial list true. Given an open branch of a tableau, such an interpretation can, in fact, be read off from the branch.[6]

1.5.3 The recipe is simple. If the propositional parameter, p, occurs at a node on the branch, assign it 1; if $\neg p$ occurs at a node on the branch, assign it 0. (If neither p nor $\neg p$ occurs in this way, it may be assigned anything one likes.)

1.5.4 For example, consider the tableau of 1.5.1 and its (bolded) leftmost open branch. Applying the recipe gives the interpretation, ν, such that $\nu(r) = 1$, and $\nu(p) = \nu(q) = 0$. It is simple to check directly that $\nu((p \supset q) \vee (r \supset q)) = 1$ and $\nu((p \vee r) \supset q) = 0$. Since p is false, $p \supset q$ is true, as is $(p \supset q) \vee (r \supset q)$. Since r is true, $p \vee r$ is true; but q is false; hence, $(p \vee r) \supset q$ is false.

[6] If one thinks of constructing a tableau as a search procedure for a counter-model, then the soundness and completeness theorems constitute, in effect, a proof that the procedure always gives the right result, that is, which *verifies* the algorithm in question.

1.5.4a Note that the tableau of 1.4.8 shows that *any* inference of the form in question is valid. That is, A, B and C can be *any* formulas. To show that an inference is invalid, we have to construct a counter-model, and this means assigning truth values to *particular* formulas. This is why the example just given uses 'p', 'q' and 'r', not 'A', 'B' and 'C'. One may say that an inference expressed using schematic letters ('A's and 'B's) is invalid, but this must mean that there are some formulas that can be substituted for these letters to make it so. Thus, we may write $A \nvDash B$, since $p \nvDash q$. But note that this does not rule out the possibility that some inferences of that form are valid, e.g., $p \vDash q \vee \neg q$.

1.5.5 As one would hope, the tableau procedure we have been looking at is sound and complete with respect to the semantic notion of consequence, i.e., if Σ is a finite set of sentences, $\Sigma \vdash A$ iff $\Sigma \models A$. That is, the search procedure really works. If there is an interpretation that makes all the formulas on the initial list true, the tableau will have an open branch which, in effect, specifies one. And if there is no such interpretation, every branch will close. These facts are not obvious. The proof is in 1.11.[7]

1.6 Conditionals

1.6.1 In the remainder of this chapter, we look at the notion of conditionality that the above, classical, semantics give us, and at its inadequacy. But first, what is a conditional?

1.6.2 Conditionals relate some proposition (the *consequent*) to some other proposition (the *antecedent*) on which, in some sense, it depends. They are expressed in English by 'if' or cognate constructions:

If the bough breaks (then) the cradle will fall.
The cradle will fall if the bough breaks.
The bough breaks only if the cradle falls.

[7] The restriction to finite Σ is due to the fact that tableaux have been defined only for finite sets of premises. It is possible to define tableaux for infinite sets of premises as well (not putting all the premises at the start, but introducing them, one by one, at regular intervals down the branches). If one does this, the soundness and completeness results generalise to arbitrary sets of premises. We will take up this matter again in Chapter 12 (Part II), where the matter assumes more significance.