(i) Assume that b^b is rational. Then this proof is easy since we can choose irrational numbers a and b to be $\sqrt{2}$ and see that a^b is just b^b which was assumed to be rational.

(ii) Assume that b^b is *ir*rational. Then we change our strategy slightly and choose a to be $\sqrt{2}^{\sqrt{2}}$. Clearly, a is irrational by the assumption of case (ii). But we know that b is irrational (this was known by the ancient Greeks; see the proof outline in the exercises). So a and b are both irrational numbers and

$$a^b = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{(\sqrt{2}\cdot\sqrt{2})} = (\sqrt{2})^2 = 2$$

is rational, where we used the law $(x^y)^z = x^{(y\cdot z)}$.

Since the two cases above are exhaustive (*either* b^b is irrational, *or* it isn't) we have proven the theorem. □

This proof is perfectly legitimate and mathematicians use arguments like that all the time. The exhaustive nature of the case analysis above rests on the use of the rule LEM, which we use to prove that either b is rational or it is not. Yet, there is something puzzling about it. Surely, we have secured the fact that there are irrational numbers a and b such that a^b is rational, but are we in a position to specify an actual pair of such numbers satisfying this theorem? More precisely, which of the pairs (a, b) above fulfils the assertion of the theorem, the pair $(\sqrt{2}, \sqrt{2})$, or the pair $(\sqrt{2}^{\sqrt{2}}, \sqrt{2})$? Our proof tells us nothing about *which* of them is the right choice; it just says that at least one of them works.

Thus, the intuitionists favour a calculus containing the introduction and elimination rules shown in Figure 1.2 and excluding the rule $\neg\neg$e and the derived rules. Intuitionistic logic turns out to have some specialised applications in computer science, such as modelling type-inference systems used in compilers or the staged execution of program code; but in this text we stick to the full so-called classical logic which includes all the rules.

1.3 Propositional logic as a formal language

In the previous section we learned about propositional atoms and how they can be used to build more complex logical formulas. We were deliberately informal about that, for our main focus was on trying to understand the precise mechanics of the natural deduction rules. However, it should have been clear that the rules we stated are valid for *any* formulas we can form, as long as they match the pattern required by the respective rule. For example, the application of the proof rule $\rightarrow$e in

1	$p \rightarrow q$	premise
2	p	premise
3	q	$\rightarrow$e 1, 2

is equally valid if we substitute p with $p \vee \neg r$ and q with $r \rightarrow p$:

1	$p \vee \neg r \rightarrow (r \rightarrow p)$	premise
2	$p \vee \neg r$	premise
3	$r \rightarrow p$	$\rightarrow$e 1, 2

This is why we expressed such rules as schemes with Greek symbols standing for generic formulas. Yet, it is time that we make precise the notion of 'any formula we may form.' Because this text concerns various logics, we will introduce in (1.3) an easy formalism for specifying well-formed formulas. In general, we need an *unbounded* supply of propositional atoms $p, q, r, \dots$, or $p_1, p_2, p_3, \dots$ You should not be too worried about the need for infinitely many such symbols. Although we may only need *finitely many* of these propositions to describe a property of a computer program successfully, we cannot specify how many such atomic propositions we will need in any concrete situation, so having infinitely many symbols at our disposal is a cheap way out. This can be compared with the potentially infinite nature of English: the number of grammatically correct English sentences is infinite, but finitely many such sentences will do in whatever situation you might be in (writing a book, attending a lecture, listening to the radio, having a dinner date, ...).

Formulas in our propositional logic should certainly be strings over the alphabet $\{p, q, r, \dots\} \cup \{p_1, p_2, p_3, \dots\} \cup \{\neg, \wedge, \vee, \rightarrow, (,)\}$. This is a trivial observation and as such is not good enough for what we are trying to capture. For example, the string $(\neg)() \vee pq \rightarrow$ is a word over that alphabet, yet, it does not seem to make a lot of sense as far as propositional logic is concerned. So what we have to define are those strings which we want to call formulas. We call such formulas *well-formed.*

Definition 1.27 The well-formed formulas of propositional logic are those which we obtain by using the construction rules below, and only those, finitely many times:

atom: Every propositional atom $p, q, r, \ldots$ and $p_1, p_2, p_3, \ldots$ is a well-formed formula.
$\neg$: If ϕ is a well-formed formula, then so is $(\neg\phi)$.
$\wedge$: If ϕ and ψ are well-formed formulas, then so is $(\phi \wedge \psi)$.
$\vee$: If ϕ and ψ are well-formed formulas, then so is $(\phi \vee \psi)$.
$\rightarrow$: If ϕ and ψ are well-formed formulas, then so is $(\phi \rightarrow \psi)$.

It is most crucial to realize that this definition is the one a computer would expect and that we did not make use of the binding priorities agreed upon in the previous section.

Convention. In this section we act as if we are a rigorous computer and we call formulas well-formed iff they can be deduced to be so using the definition above.

Further, note that the condition 'and only those' in the definition above rules out the possibility of any other means of establishing that formulas are well-formed. Inductive definitions, like the one of well-formed propositional logic formulas above, are so frequent that they are often given by a defining grammar in Backus Naur form (BNF). In that form, the above definition reads more compactly as

$$\phi ::= p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \tag{1.3}$$

where p stands for any atomic proposition and each occurrence of ϕ to the right of $::=$ stands for any already constructed formula.

So how can we show that a string is a well-formed formula? For example, how do we answer this for ϕ being

$$(((\neg p) \wedge q) \rightarrow (p \wedge (q \vee (\neg r)))) \ ? \tag{1.4}$$

Such reasoning is greatly facilitated by the fact that the grammar in (1.3) satisfies the *inversion principle*, which means that we can invert the process of building formulas: although the grammar rules allow for five different ways of constructing more complex formulas – the five clauses in (1.3) – there is always a unique clause which was used last. For the formula above, this last operation was an application of the fifth clause, for ϕ is an implication with the assumption $((\neg p) \wedge q)$ and conclusion $(p \wedge (q \vee (\neg r)))$. By applying the inversion principle to the assumption, we see that it is a conjunction of $(\neg p)$ and q. The former has been constructed using the second clause and is well-formed since p is well-formed by the first clause in (1.3). The latter is well-formed for the same reason. Similarly, we can apply the inversion

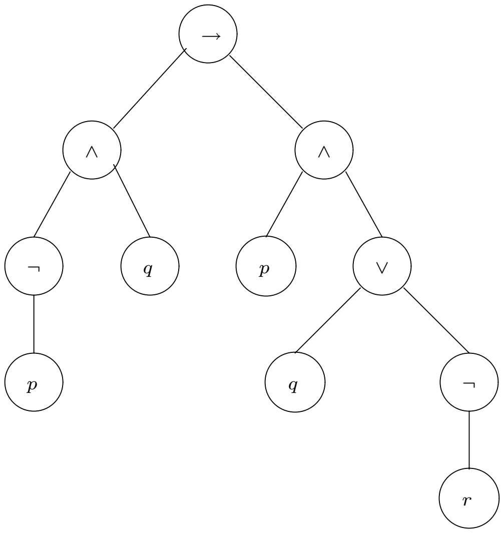

Figure 1.3. A parse tree representing a well-formed formula.

principle to the conclusion $(p \wedge (q \vee (\neg r)))$, inferring that it is indeed well-formed. In summary, the formula in (1.4) is well-formed.

For us humans, dealing with brackets is a tedious task. The reason we need them is that formulas really have a tree-like structure, although we prefer to represent them in a linear way. In Figure 1.3 you can see the parse tree[7] of the well-formed formula ϕ in (1.4). Note how brackets become unnecessary in this parse tree since the paths and the branching structure of this tree remove any possible ambiguity in interpreting ϕ. In representing ϕ as a linear string, the branching structure of the tree is retained by the insertion of brackets as done in the definition of well-formed formulas.

So how would you go about showing that a string of symbols ψ is *not* well-formed? At first sight, this is a bit trickier since we somehow have to make sure that ψ could not have been obtained by *any* sequence of construction rules. Let us look at the formula $(\neg)() \vee pq \rightarrow$ from above. We can decide this matter by being very observant. The string $(\neg)() \vee pq \rightarrow$ contains $\neg)$ and $\neg$ cannot be the rightmost symbol of a well-formed formula (check all the rules to verify this claim!); but the only time we can put a ')' to the right of something is if that something is a well-formed formula (again, check all the rules to see that this is so). Thus, $(\neg)() \vee pq \rightarrow$ is *not* well-formed.

Probably the easiest way to verify whether some formula ϕ is well-formed is by trying to draw its parse tree. In this way, you can verify that the

[7] We will use this name without explaining it any further and are confident that you will understand its meaning through the examples.

formula in (1.4) is well-formed. In Figure 1.3 we see that its parse tree has $\rightarrow$ as its root, expressing that the formula is, at its top level, an implication. Using the grammar clause for implication, it suffices to show that the left and right subtrees of this root node are well-formed. That is, we proceed in a top-down fashion and, in this case, successfully. Note that the parse trees of well-formed formulas have either an atom as root (and then this is all there is in the tree), or the root contains $\neg$, $\vee$, $\wedge$ or $\rightarrow$. In the case of $\neg$ there is only *one* subtree coming out of the root. In the cases $\wedge$, $\vee$ or $\rightarrow$ we must have *two* subtrees, each of which must behave as just described; this is another example of an *inductive* definition.

Thinking in terms of trees will help you understand standard notions in logic, for example, the concept of a *subformula*. Given the well-formed formula ϕ above, its subformulas are just the ones that correspond to the subtrees of its parse tree in Figure 1.3. So we can list all its leaves p, q (occurring twice), and r, then $(\neg p)$ and $((\neg p) \wedge q)$ on the left subtree of $\rightarrow$ and $(\neg r)$, $(q \vee (\neg r))$ and $((p \wedge (q \vee (\neg p))))$ on the right subtree of $\rightarrow$. The whole tree is a subtree of itself as well. So we can list all nine subformulas of ϕ as

$$
\begin{array}{l}
p \\
q \\
r \\
(\neg p) \\
((\neg p) \wedge q) \\
(\neg r) \\
(q \vee (\neg r)) \\
((p \wedge (q \vee (\neg r)))) \\
(((\neg p) \wedge q) \rightarrow (p \wedge (q \vee (\neg r)))).
\end{array}
$$

Let us consider the tree in Figure 1.4. Why does it represent a well-formed formula? All its leaves are propositional atoms (p twice, q and r), all branching nodes are logical connectives ($\neg$ twice, $\wedge$, $\vee$ and $\rightarrow$) and the numbers of subtrees are correct in all those cases (one subtree for a $\neg$ node and two subtrees for all other non-leaf nodes). How do we obtain the linear representation of this formula? If we ignore brackets, then we are seeking nothing but the *in-order* representation of this tree as a list[8]. The resulting well-formed formula is $((\neg(p \vee (q \rightarrow (\neg p)))) \wedge r)$.

[8] The other common ways of flattening trees to lists are *preordering* and *postordering*. See any text on binary trees as data structures for further details.

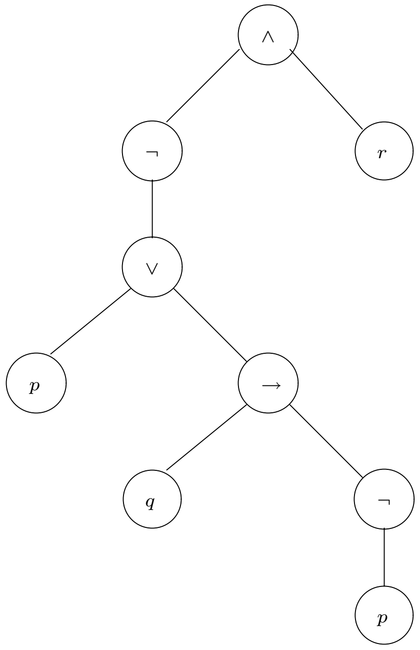

Figure 1.4. Given: a tree; wanted: its linear representation as a logical formula.

The tree in Figure 1.21 on page 82, however, does *not* represent a well-formed formula for two reasons. First, the leaf $\wedge$ (and a similar argument applies to the leaf $\neg$), the left subtree of the node $\to$, is not a propositional atom. This could be fixed by saying that we decided to leave the left and right subtree of that node unspecified and that we are willing to provide those now. However, the second reason is fatal. The p node is not a leaf since it has a subtree, the node $\neg$. This cannot make sense if we think of the entire tree as some logical formula. So this tree does not represent a well-formed logical formula.

1.4 Semantics of propositional logic

1.4.1 The meaning of logical connectives

In the second section of this chapter, we developed a calculus of reasoning which could verify that sequents of the form $\phi_1, \phi_2, \ldots, \phi_n \vdash \psi$ are valid, which means: from the premises ϕ_1, ϕ_2, …, ϕ_n, we may conclude ψ.

In this section we give another account of this relationship between the premises ϕ_1, ϕ_2, …, ϕ_n and the conclusion ψ. To contrast with the sequent