This is from Chapter 2 of textbook “Logic in Action” (www.logicinaction.org)

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
$\rightarrow$	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	$\neg(k \vee s) \rightarrow a$

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) $\rightarrow$ (he has an Ace),
not (he has a Knight or a Spade) $\rightarrow$ (he has an Ace),
$\neg$ (he has a Knight or a Spade) $\rightarrow$ (he has an Ace),
$\neg$ ((he has a Knight) or (he has a Spade)) $\rightarrow$ (he has an Ace),
$\neg$ ((he has a Knight) $\vee$ (he has a Spade)) $\rightarrow$ (he has an Ace),
$\neg(k \vee s) \rightarrow a$

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

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

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 $w \rightarrow s$. 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.

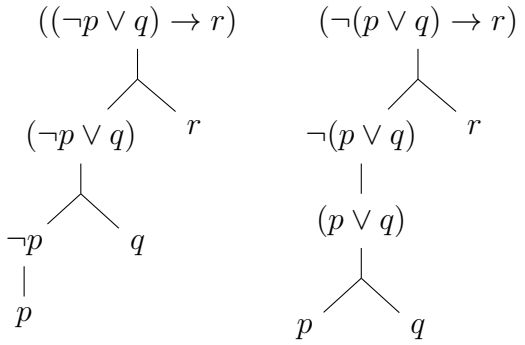
Example 2.4 The formula $((\neg p \vee q) \rightarrow r)$ is created stepwise from proposition letters p, q, r by applying the following construction rules successively:

(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 $((\neg p \vee q) \rightarrow r)$

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 $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \rightarrow \varphi_2)$ 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 $p \in P$.

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

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:

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

Exercise 2.8 Construct trees for the following formulas:

- $(p \wedge q) \rightarrow \neg q$
- $q \wedge r \wedge s \wedge t$ (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.